공학박사학위논문

Synthesizing and Editing Human Motion from Sparse User Inputs

적은 수의 사용자 입력으로부터
인간 동작의 합성 및 편집

2014년 8월

서울대학교 대학원
전기·컴퓨터공학부
김 종 민

Synthesizing and Editing Human Motion from Sparse User Inputs

적은 수의 사용자 입력으로부터

인간 동작의 합성 및 편집

지도교수 이 제 희

이 논문을 공학 박사 학위논문으로 제출함

2014년 8월

서울대학교 대학원

전기·컴퓨터공학부

김 종 민

김종민의 공학박사 학위논문을 인준함

2014년 8월

위 원 장	김 명 수
부위원장	이 제 희
위 원	장 병 탁
위 원	서 진 욱
위 원	권 태 수

Abstract

An ideal 3D character animation system can easily synthesize and edit human motion and also will provide an efficient user interface for an animator. However, despite advancements of animation systems, building effective systems for synthesizing and editing realistic human motion still remains a difficult problem. In the case of a single character, the human body is a significantly complex structure because it consists of as many as hundreds of degrees of freedom. An animator should manually adjust many joints of the human body from user inputs. In a crowd scene, many individuals in a human crowd have to respond to user inputs when an animator wants a given crowd to fit a new environment. The main goal of this thesis is to improve interactions between a user and an animation system.

As 3D character animation systems are usually driven by low-dimensional inputs, there is no method for a user to directly generate a high-dimensional character animation. To address this problem, we propose a data-driven mapping model that is built by motion data obtained from a full-body motion capture system, crowd simulation, and data-driven motion synthesis algorithm. With the data-driven mapping model in hand, we can transform low-dimensional user inputs into character animation because motion data help to infer missing parts of system inputs. As motion capture data have many details and provide realism of the movement of a human, it is easier to generate a realistic character animation than without motion capture data.

To demonstrate the generality and strengths of our approach, we developed two animation systems that allow the user to synthesize a single character animation in realtime and edit crowd animation via low-dimensional user inputs interactively. The first system entails controlling a virtual avatar using a small set of three-dimensional (3D) motion sensors. The second system manipulates large-scale crowd animation that consists of hundreds of characters with a small number of user constraints. Examples show that our system is much less laborious and time-consuming than previous animation systems, and thus is much more suitable for a user to generate desired character animation.

Keywords: Computer Graphics, Human Motion, Data-driven Animation, Performance Animation, Interactive Editing, Machine Learning, Numerical Optimization

Student Number: 2007-20955

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

Representing human motions has long been an important part of human activities because it can transfer a story to other humans. Cave paintings made thousands years ago might have been utilized as a way of communicating with other humans or for religious purposes. Paintings often describe either human action or the results of a human action. However, a static painting has a limitation in terms of conveying a large amount of story because of its expression in limited space. To resolve this problem, mankind came up with the idea of generating several series of images from paintings as an efficient means of storytelling. In 1900, *Enchanted Drawing*, the first film including animated human sequences was introduced by J. Stuart Blackton, who is considered the father of animation. Animation feature films were later introduced in the 1930s by Walt Disney Studios. Walt Disney Studios pro-

duced these feature films based on sequences of hand-drawn two-dimensional (2D) cartoon images, including Snow White (1937), Dumbo (1941), and Bambi (1942). These well hand-drawn films that include interesting characters and environments in the scene allow the audience to take part in the story. The characters were actually extremely appealing to the audience of the time and even to some audiences today. However, the quality of character animation was not sufficient to efficiently convey story lines. After three-dimensional (3D) computer animation research matured, 3D character models can be generated efficiently. The 3D character motion can be obtained by setting keyframes and making the computer produce in between keyframes. Motion capture systems also help animators to create realistic human animation scenes, because they can faithfully capture human movement. With such 3D computer systems, well-trained animators can create complex and consistent 3D character animation. Many 3D characters generated from advanced computer animation techniques have played an important role to pursue contemporary story telling and have led to commercially successful films such as Toy Story (1995), Cars (2006), and Frozen (2013).

Animated human characters have now become part of our everyday life as we are regularly exposed to media such as animated films, video games, and interactive applications. One of the greatest requirements of such media is realism of animated human. Makers of current animated films are particularly interested in realistic virtual characters to deeply evoke empathy and sympathy in the audience. To achieve this goal, it is necessary to generate not only natural appearance, but also seamless movement of a virtual character. In many video games, a gamer desires to interactively control a game character for all required tasks. The video games must continuously handle user inputs to control game characters with

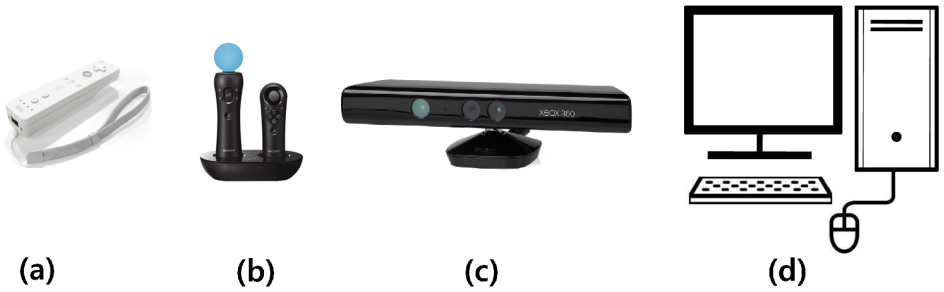

Figure 1.1: *Examples of user interfaces: (a) Nintendo's Wiimote; (b) Sony's Move; (c) Microsoft's Kinect; (d) Typical keyboard and mouse interface of personal computer.*

sufficient quality. If the animated humans in the media have deficiencies in terms of subtle artifacts or small delays, the media will fail in terms of entertainment and will likely be unsuccessful commercially. Therefore, animated humans should be as realistic as possible in order to immerse viewers and players in commercial movies and video games.

There are two main difficulties to generate realistic human animation. First, as many people live their entire lives while observing other's movements, they have a high sensitivity to identify subtle artifacts in human animation. As a result, people can easily judge whether their movements are realistic or not; creating good-quality human motion is thus a critical requirement. Second, the human body is a significantly complex structure, having as many as hundreds of degrees of freedom. To control the human body precisely, an animator should manually manipulate many joints on the body. During animation and game production, the animators engage exhaustively in laborious trial-and-errors when trying to obtain the desired human animation.

The challenge of producing realistic character motion arises not only for a single character, but also a human crowd, which consists of many characters. A common observation on

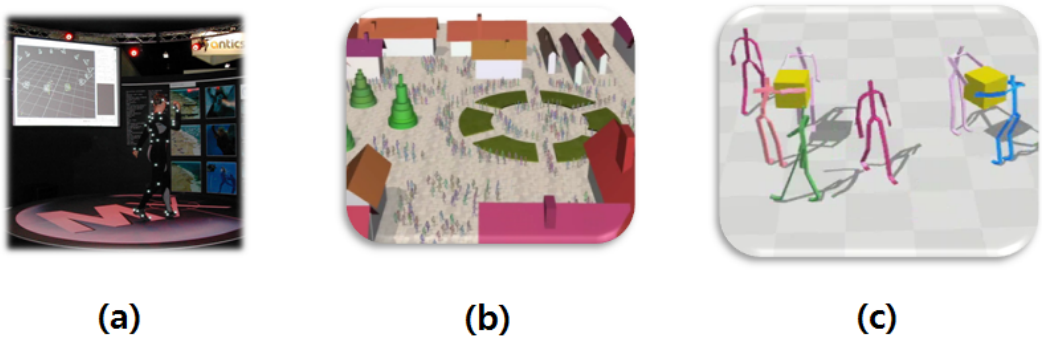

Figure 1.2: *Prior knowledge from various sources: (a) motion capture data from Vicon motion capture system; (b) crowd simulation data from [30]; (c) motion data from data-driven motion synthesis algorithm from [33].*

human crowds is that they are not just an array of individuals. Each individual in a human crowd is highly dependent on the other individuals in terms of physiological and social factors. In general, many interactions exist between individuals in a human crowd and they can sometimes act collaboratively or adversarially; example behaviors include hand shaking, carrying something heavy, and pushing each other. During the animation process, these interactions cannot be easily created because they should be carefully coordinated in both space and time. Many individuals in a human crowd also have to promptly respond to user intention and change their locations according to a new environment. As the number of individuals and interactions between them grow, an animator will consume increasingly greater time to fit the crowd animation in the new environment. A character animation system without an excessive amount of user interaction is thus highly demanded for an animator to create complex human crowd scenes.

Numerous user interfaces for creating character animation are available, such as button commands, keystrokes, accelerometers, depth cameras, and so on (see Figure 1.1). As an-

imation systems are usually based on low-dimensional user inputs from user interfaces, there is no method for a user to directly generate a high-dimensional character animation. Therefore, an additional modality that provides a user with the capability to control human motions through such traditional interfaces is essential for reducing the user's effort and required time.

1.2 Approach

This thesis explores approaches for creating high-quality and realistic character motions interactively by various types of user inputs. The information acquired from user input is often low-dimensional, and thus, in general, high-dimensional realistic character motions cannot be inferred without prior knowledge. The missing information can be inferred by employing prior knowledge, which can be extracted from a motion capture database. We set up prior knowledge from various sources including a full-body motion capture system, crowd simulation data, and a data-driven motion synthesis algorithm (see Figure 1.2). We then build a *mapping model* that transforms low-dimensional user inputs into realistic character motions. We can infer both single and multi-character animation from the obtained data-driven mapping model (see Figure 1.3).

The mapping model established from the prior knowledge has to satisfy two key components. First, it should preserve the characteristics of prior knowledge properly. Incorporating the prior knowledge and mapping model may lead to degradation of human motion quality because of irrelevant information in the prior knowledge. The mapping model should be carefully established by extracting meaningful information from prior knowledge

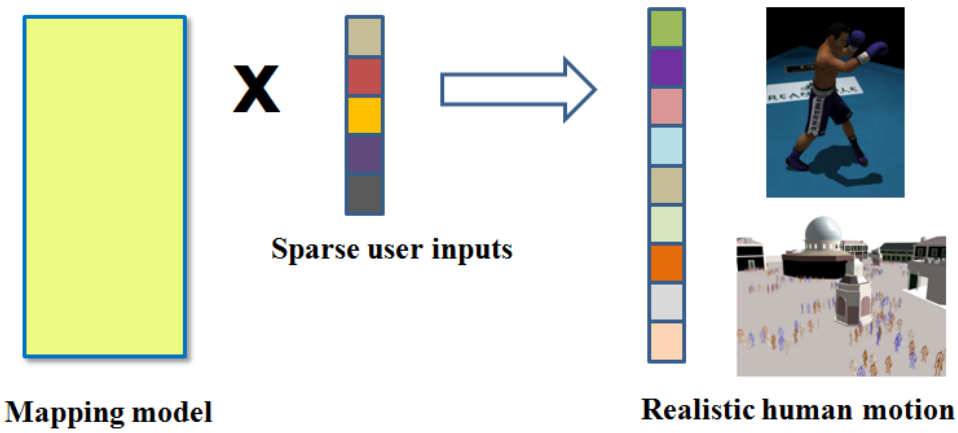

Figure 1.3: *With prior knowledge in hand, we can construct a data-driven mapping model that transforms sparse user inputs to realistic human motion. Realistic motion can be obtained by matrix-vector multiplication. A data-driven mapping model can be represented as a matrix while the input vector is a low-dimensional user input. The output vector can be represented as the realistic human motion, either for a single character animation or crowd animation.*

to achieve higher accuracy. Second, the mapping model has to satisfy the user constraints precisely. Temporal and spatial user constraints on the user's intention should be satisfied with preservation of the traits of prior knowledge and it should be possible to extend various styles of human movement.

To demonstrate the generality and strengths of our approach, we developed two animation systems that allow the user to generate either single or multi-character animation via sparse user inputs. The first system involves controlling a virtual avatar using sparse 3D motion sensors. The second interactively manipulates a large-scale crowd animation with sparse user constraints. We believe that high-quality character animation can be created with our

Figure 1.4: *By utilizing our performance animation system, the boxing motion of the user with sparse 3D motion sensors is transferred to the virtual character.*

systems. Furthermore, systems are less laborious and time-consuming than previous animation systems, and are suitable for novice users to generate desired character animation.

Performance Animation Using Sparse 3D Motion Sensors. Recent advancements in gaming interfaces such as Nintendo's Wiimote and Sony's Move have improved the user's gaming experience by allowing various performance animation applications in virtual environments (see Figure 1.1). Users can play virtual sports, dance, and do other specific activities using their body as a controller. Such applications usually depend on sparse input data from corresponding body parts (e.g., two acceleration sensors on each hand, tracking one vision based marker), and cannot accurately reproduce the full-body pose of a performer. Despite the various potential applications, reconstruction of an accurate full-body pose in a low-dimensional input setting remains a challenging problem.

We explored an approach that uses sparse 3D motion sensors to create a practical performance animation system (see Figure 1.4). The signals from 3D motion sensors are generally noisy and sparse, and thus cannot be directly used for precise control of high-dimensional characters. We employ a statistical data-driven approach, kernel CCA-based regression [15], which fully explains the nonlinear relationship between 3D motion sensors

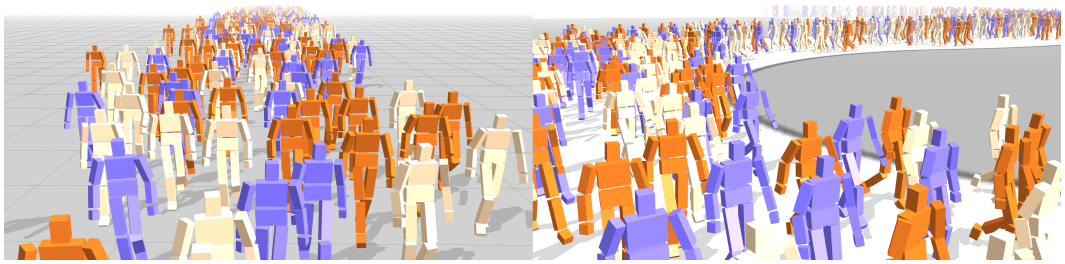

Figure 1.5: *Many pedestrians walk straight in the crowd animation (left). We interactively manipulate the crowd animation to follow an s-curve (right).*

and character pose. Motion capture data are utilized as prior knowledge. From the captured motion data, we build a data-driven mapping model that transforms low-dimensional input signals into high-quality human motions.

Interactive Manipulation of Large-Scale Crowd Animation. Many recent feature films and video games often show spectacular scenes with large crowds of characters. Crowd animation is used to show a variety of interesting events. The characters are, at times, arranged tightly into small spaces. They sometimes interact with each other collaboratively or adversarially. There is a large array of crowd simulation algorithms that can generate a crowd of animated characters. These algorithms often have a set of tunable parameters. An animator adjusts the parameters to control how the crowd animation should look. The production of crowd simulation is a trial-and-error process of running multiple simulations with different parameters.

In this thesis, we present a novel crowd editing system for manipulating large-scale crowd animation, interactively (see Figure 1.5). The benefits of our editing method are manifold. From the user interface point of view, we provide coherent control over multiple character motions such that the motions are manipulated in a coordinated manner. We also utilize a

manipulation mapping model that combines Mean Value Coordinates (MVC) [16, 21] and as-rigid-as possible deformation [68] to manipulate the crowd animation smoothly. Our system also allows for locality of control. The animator can manipulate a small portion of the crowd animation. Our method does not impose any limitation on human motion. Arbitrary types of human motion can be incorporated in the crowd animation, including even interpersonal interaction with physical contact. From a performance point of view, our manipulation mapping model serves as a reduced model of the animation; it can thus be manipulated efficiently based on two-level hierarchical computation. Our method is faster than the previous methods by orders of magnitude. The performance gain is even more substantial for a larger number of characters in a crowd.

1.3 Thesis Overview

The remainder of the thesis is organized as follows. Chapter 2 presents the background of performance animation methods and the various attempts to generate a large-scale crowd animation. Chapter 3 describes how to control a single character animation using sparse 3D motion sensors. Chapter 4 shows how to manipulate a large-scale crowd animation in both space and time domains interactively. Finally, Chapter 5 concludes the thesis and describes possible future research directions.

Chapter 2

Background

In this chapter, we describe the previous approaches and background required to understand data-driven mapping models for synthesizing and editing human motion. In the first section, we introduce a wide range of studies related to performance animation. We review many types of interfaces and various statistical approaches with extracting meaningful animation parameters to reconstruct an accurate character animation. In the second section, we introduce research related to generate a crowd animation. We review the existing studies based on crowd simulation, motion editing, and geometry deformation to manipulate crowd animation.

2.1 Performance Animation

2.1.1 Performance-based Interfaces for Character Animation

Reconstructing human motions of characters is an important issue in computer animation and has long been studied. Researchers have obtained raw human motion using magnets, optical devices, and video cameras. The Vicon motion capture system is used to record the 3D movement of a user who wears a skin-tight garment with dozens of attachable optical markers. Badler et al. [3] employed four magnetic sensors and an inverse kinematics method to control the standing pose of a full-body character. Shin et al. [66] mapped the actions of a performer to a virtual character using a magnetic motion capture system. Yin and Pai [77] proposed an animation system that controls avatars with a foot pressure sensor pad. However, a problem arises when attempting to generate accurate upper-body motions with restricted foot pressure information. Lee et al. [40] used a vision-based technique to capture full-body motion from a single camera. Chai and Hodgins [9] reconstructed human motion using a small set of optical markers and two video cameras. Low-cost hardware to capture human motion was recently developed for game interfaces. Wiimote controller by Nintendo is a popular interface that uses an accelerometer and optical sensors to allow the system to recognize human gestures. Sony's play Station Move is another motion-sensing controller that tracks the three-dimensional position of the controller using inertial sensors. Microsoft Kinect can generate 3D avatars without attachable hardware. While this hardware can provide an exciting game experience in a constrained environment, it is not applicable for general-purpose retargeting. Our sensor-based performance animation system shares the affordable cost advantage of these input devices while remaining applicable

to a wider range of situations. Slyper and Hodgins [67] created a system that searches a motion database with low-cost accelerometers and then plays the best match of motion sequences from a database. Tautges et al. [72] introduced a performance animation system that generates full-body character animation using four accelerometers. Additionally, Liu et al. [50] introduced an online animation system with a small number of inertial sensors as a performance animation interface. Our approach has an environment setup similar to that in [50]; however, we suggest an alternative data-driven approach for performance animation. We utilize kernel CCA-based regression concept, introduced in [15], which achieves strong interpolation and extrapolation capabilities by applying a kernel trick method to CCA-based regression scheme presented in [22].

2.1.2 Statistical Models for Motion Synthesis

Many motion synthesis studies have utilized statistical analyses from motion capture data. Brand and Hertzmann [7] employed a Gaussian mixture model to estimate the distribution of given motion data, inferring the desired poses by computing several parameters. Shin and Lee [65] proposed an interactive motion synthesis framework for use in low-dimensional space. Liu et al. [49] exploited a probabilistic PCA (Principle Components Analysis) to predict full-body human motions from a set of principle markers. Chai and Hodgins [8] constructed a statistical dynamic model to generate motion sequences satisfying given user constraints. Wei and Chai [75] introduced a system that interactively controls a 3D human character using a factor analysis with millions of motion examples. Our work is motivated by these successful statistical approaches. We consider the idea of constructing a nonlin-

ear statistical model as presented in earlier work [15] to handle the nonlinear relationship between 3D motion sensors and the human pose. Motion reproduction based on statistical models encounters difficulties when the size of the motion database becomes bigger. Grochow et al. [18] applied the GPLVM (Gaussian Process Latent Variable Model) using motion examples and a set of kinematic constraints to create realistic human motion. Although their approach can generate faithful animation results, it has difficulties when synthesizing high-quality animation with heterogeneous motions due to intrinsic problems associated with global modeling approaches. For faithful local modeling, an efficient retrieval method is required. Many researchers have studied local modeling approaches to address this problem.

2.1.3 Retrieval of Motion Capture Data

To construct a faithful local model, efficient retrieval of motion from the database is required. Müller [55] introduced the content-based motion retrieval method. Chai and Hodgins [9] proposed the approach of searching for similar pose examples in a heterogeneous motion database using a precomputed neighborhood graph, and generates the resulting pose using a local model at every frame. In another work [72], Tautges et al. suggested a more efficient nearest-neighbor search method that describes temporal coherence and a large scale motion database. Their approach is based on an earlier study [36] that proposed a kd-tree-based search method to find exact neighborhood poses in a large motion database. For the retrieval of motions, we incorporate the kd-tree-based search and temporal features such as velocity and acceleration. With this approach, we retrieve similar pose examples to the

performer's movement and use them to build a nonlinear local model for human motion reconstruction.

2.2 Crowd Animation

2.2.1 Crowd Simulation

Many researchers have explored the generation of realistic crowd animations. These include agent-based decision making methods [62, 56, 61, 60, 19], flow control based on a velocity field [10, 73], and a mixture of these two approaches [57]. Data-driven approaches have also been studied to create realistic crowd animation. Two-dimensional motion data of a group of people captured from a bird's-eye view are used to mimic the complex behavior of a real crowd [42, 44, 45, 30]. While many methods have been suggested for convincing crowd generation, a few methods [37, 33, 28] are available for the direct editing of existing crowd animations.

2.2.2 Motion Editing

Editing existing human motion to fit a new constraint has long been studied and is an important issue in character animation. Computing additional low-frequency motions with the given constraints [17, 41] has been widely employed as it provides a desired motion with original characteristics. Based on a set of motion examples, the blending [34, 54] and rearrangement [40, 35, 1, 63] of motions have been employed to synthesize new motion

sequences. Interactive manipulation of multi-character animation was first addressed by Kwon et al. [37]. They used a graph structure to model the spatio-temporal group behavior of pedestrians and employed a mesh editing algorithm to manipulate the animation interactively. Kim et al. [33] and Ho et al. [20] extended the manipulation method to handle a wider range of group behavior with complex interactions. These previous methods commonly transform motion editing into a constraint-solving problem. The size of the constraint-solving problem scales rapidly with respect to the number of characters and the duration of the animation. We present a novel interactive method to address the constraint-solving problem. Our approach is versatile, efficient, easy-to-use, and the interactive techniques facilitate the user's work process.

2.2.3 Geometry Deformation

The efficiency of our method mainly comes from the hybrid use of two geometry deformation approaches: cage-based deformation and surface-based deformation. In cage-based deformation, the target geometry is represented as linear combinations of an external cage that encloses the target. Manipulating a boundary vertex of the cage results in smooth deformation of the target geometry. Mean Value Coordinates (MVC) [16, 21] is a simple and popular way of parameterizing the interior of a cage. Positive MVC [46], harmonic coordinates [29], green coordinates [47], and bounded biharmonic weights [27, 26] are more sophisticated alternatives. Surface-based deformation refers to a class of energy-minimization techniques that allow for direct manipulation of a geometric object while preserving its original shape as much as possible [69, 48, 24, 68, 5]. Surface-based deformation methods

usually require a large system of linear/non-linear equations to be solved given a high-resolution target geometry. Hybrid deformation methods have been explored to achieve both affordable performance and the convenience of direct manipulation [23, 76, 71, 4]. We refer readers to [12] for comprehensive survey on both approaches.

Chapter 3

Realtime Performance Animation Using Sparse 3D Motion Sensors

In this chapter, we presents a real-time performance animation system that reproduces full-body character animation based on sparse 3D motion sensors on a performer. Producing faithful character animation from this setting is a mathematically ill-posed problem because input data from the sensors are not sufficient to determine the full degrees of freedom of a character. Given the input data from 3D motion sensors, we select similar poses from a motion database and build an online local model that transforms the low-dimensional input signal into a high-dimensional character pose. A regression method based on kernel CCA (Canonical Correlation Analysis) is employed because it effectively handles a wide variety of motions. Examples show that various human motions are naturally reproduced by the proposed method.

3.1 Overview

Recent advancements in gaming interfaces such as Nintendo's Wiimote and Sony's Move have improved the user's gaming experience by allowing various performance animation applications in a virtual environment. Users can play virtual sports, dance, and engage in other activities using their bodies as a controller. Such applications usually depend on the input data from sparse body parts (e.g., two acceleration sensors on each hand, tracking of a vision based marker) and therefore cannot accurately reproduce the full-body pose of a performer. Despite the various potential applications of human motion based animation, the reconstruction of an accurate full-body pose from a sparse input setting remains a challenging problem.

We present real-time performance animation system that can be used to realize full-body animation based on sparse 3D motion sensors attached onto a performer's body. We select 3D motion sensors as an input device because they are free from occlusion, a typical problem associated with optical motion capture systems. However, it is difficult to implement a performance animation system using a sparse set of 3D motion sensors because the system input is not sufficient to determine a high-dimensional full-body character pose. We therefore consider a data-driven approach using motion capture data to predict full-body motion.

The concept of utilizing motion capture data as prior knowledge has been widely exploited in performance animation systems to transform low-dimensional data into high-dimensional character animation. Previously, data-driven online animation systems were addressed in [9, 50]. They set up an online linear local model for every frame using a

pre-captured motion database to handle various motions. Although previous methods generated convincing results, reconstructing accurate motion sequences based on linear statistical models is limited by the intrinsic nonlinear relationship between 3D motion sensors and character animation.

We employ a nonlinear statistical model, kernel CCA-based regression [15], which fully explains the nonlinear relationship between 3D motion sensors and character pose. Our method also has an advantage in terms of its ability to express a wide range of character animation with an insufficient motion dataset. Kernel CCA-based regression learns the nonlinear structures of high-dimensional motion data and forms a connection between the user's input from 3D motion sensors and the full-body character pose. At run-time, the system promptly searches for motion examples that are similar to the input motion in the motion database. The collected examples are used to train kernel CCA-based regression as an online local model and the system input then determines the character pose using the learned model.

The resulting character animation is shown to be a convincing recreation of the performer's input motions with examples of boxing, walking, and table tennis. We evaluate the performance of our system with different sensor setups and query metrics to generate the most desired results. We also show that our performance animation system can generate more accurate character animation than previous methods.

3.2 System Overview

Our system uses a motion database from five 3D motion sensors to reconstruct full-body character poses. As the input data from the sensors do not provide enough information to restore an accurate full-body character, we also employ the motion database. Based on existing motion data, we build a prediction model that produces a full-body character pose.

Figure 3.1 shows the overall work flow of the proposed performance animation system. Given the input data from 3D sensors, calibration is performed to obtain consistent input data with respect to the motion capture database. Our system rapidly searches for pose examples that are similar to the given system input and builds kernel CCA-based regression model. We then estimate the whole-body character pose with the learned model. A post-processing step reduces noise in the retargeting result.

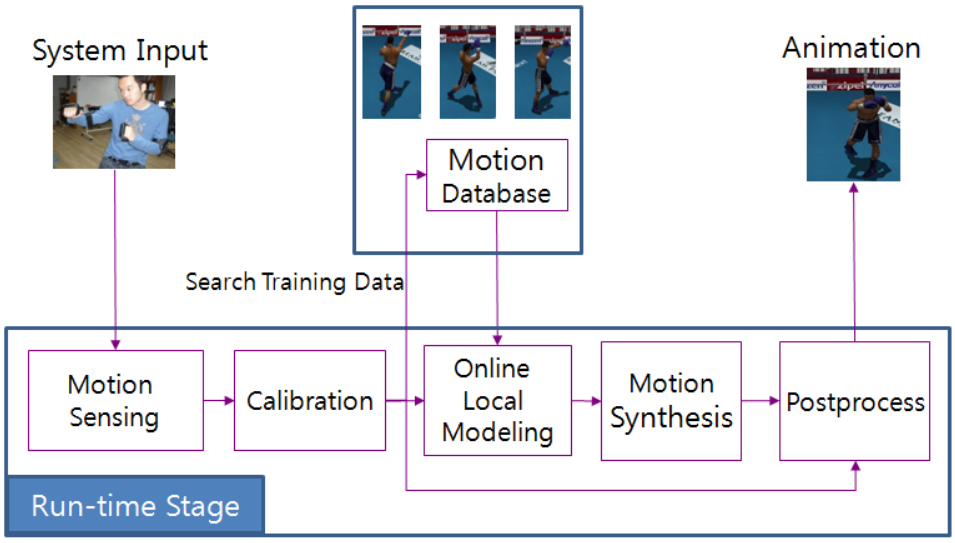

Figure 3.1: *Overall work flow of our performance animation.*

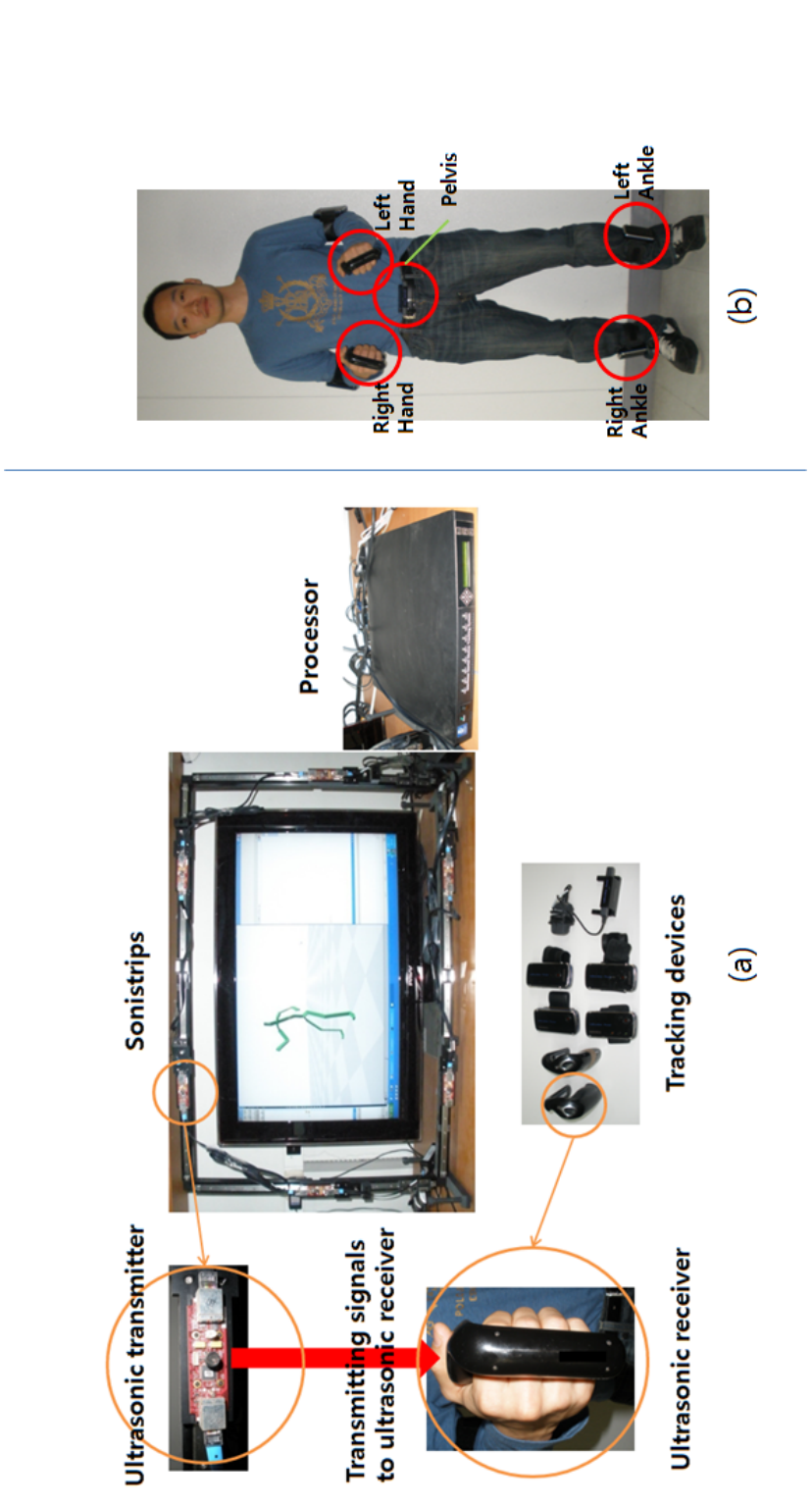

Figure 3.2: *3D motion sensor setups: (a) each transmitter transmits a signal to the receiver (3D motion sensor); (b) 3D motion sensors are attached to the hands, pelvis, and ankles of the performer.*

3.3 Sensor Data and Calibration

We use the InterSense IS-900 system [25], which offers robust real-time tracking. Figure 3.2 shows our tracking system setup. Each tracking device (3D motion sensor) has a gyroscope and an accelerometer to measure the angular and acceleration rate values, respectively. Using the measured values, the processor computes 6-DOF tracking data, which are employed in the subsequent stages. The performer wears sparse 3D motion sensors on the target joints, which are the pelvis, right hand, left hand, right ankle, and left ankle in this study (Figure 3.2), and is allowed to move freely in front of the IS-900 processor within its maximum sensing area of 20 m^2. The system provides absolute positional accuracy of 2-5mm and orientation accuracy of 1°(yaw) and 0.4°(pitch, roll).

The 6-DOF tracking data from the 3D motion sensors must be transformed from the world coordinate frame to the virtual coordinate frame. At the beginning of every capture instance, we ask the performer to begin with a standard "T-pose" to calibrate the information from the 3D motion sensors. We transform the positions and orientations of the i-th 3D motion sensor in world coordinates, $\mathbf{p}_s^i$ and $\mathbf{q}_s^i$, into their corresponding values in virtual coordinates, $\tilde{\mathbf{p}}_s^i$ and $\tilde{\mathbf{q}}_s^i$, as follows:

$$\tilde{\mathbf{p}}_s^i = \mathbf{T}_p^i \mathbf{p}_s^i + \mathbf{b}^i, \;\; \tilde{\mathbf{q}}_s^i = \mathbf{T}_q^i \mathbf{q}_s^i, \tag{3.1}$$

where $\mathbf{T}_p^i \in \mathbb{R}^{3\times3}$ and $\mathbf{b}^i \in \mathbb{R}^{3\times1}$ denote the linear transformation matrix and translational component of the i-th 3D motion sensor, respectively. Here, $\mathbf{T}_q^i$ denotes the transformation of the quaternion to orient the sensor. More precise calibration can be achieved with a nonlinear mapping model, although this significantly increases the overall number of calibration parameters. In contrast to the nonlinear model, our calibration process is easy-

to-implement and showed a reasonable level of accuracy in experiments. With ten different subjects, we observed that orientation errors are negligibly small, within 2°, and position errors are within 3 cm.

3.4 Motion Synthesis

In this section, we describe the construction process of the online local model. Kernel CCA-based regression serves as the online local model due to its extrapolation capacity. Character animation is generated using the learned model and temporal noise is reduced in the post-processing stage.

3.4.1 Online Local Model

We suggest an online local modeling approach that can produce various types of motion sequences, i.e., types that cannot be easily generated by a global modeling approach. For a global analysis, the connection between 3D motion sensors and a certain pose is determined by processing the entire set of heterogeneous motion capture data. There is an important limitation, however, when using this method. Prediction errors associated with regression can increase dramatically such that the system can scarcely generate the desired motion sequences.

To build a local model at every frame, we first choose the training data for the model. In the motion database, the k-nearest poses of the current input from the pose are chosen as the training data. We employ the position, orientation, velocity, angular velocity, acceleration,

and angular acceleration of sensors for the following query metrics:

$$c_p \sum_{i=1}^{n} \|\tilde{\mathbf{p}}_s^i - \mathbf{p}_m^i\|^2 + c_q \sum_{i=1}^{n} \|\tilde{\mathbf{q}}_s^i - \mathbf{q}_m^i\|^2 + c_v \sum_{i=1}^{n} \|\tilde{\mathbf{v}}_s^i - \mathbf{v}_m^i\|^2 + \\ c_w \sum_{i=1}^{n} \|\tilde{\mathbf{w}}_s^i - \mathbf{w}_m^i\|^2 + c_d \sum_{i=1}^{n} \|\tilde{\mathbf{d}}_s^i - \mathbf{d}_m^i\|^2 + c_e \sum_{i=1}^{n} \|\tilde{\mathbf{e}}_s^i - \mathbf{e}_m^i\|^2, \tag{3.2}$$

where $\tilde{\mathbf{v}}_s^i$, $\tilde{\mathbf{w}}_s^i$, $\tilde{\mathbf{d}}_s^i$, and $\tilde{\mathbf{e}}_s^i$ represent the velocity, angular velocity, acceleration, and angular acceleration of the i-th 3D motion sensor in the calibration pose, respectively. Here, $\mathbf{v}_m^i$,$\mathbf{w}_m^i$,$\mathbf{d}_m^i$, and $\mathbf{e}_m^i$ represent the same types of data pertaining to the i-th end-effector of a pose in the motion database, respectively. c_p, c_q, c_v, c_w, c_d, and c_e are the weight values for each respective term. In Equation (2), temporal features (e.g., the velocity, acceleration, angular velocity, and angular acceleration) are incorporated to better choose similar pose examples to the performer's movement. For example, we can correctly identify the direction of swing motion of the performer when playing table tennis by considering the temporal features.

As the total number of motions in the motion database grows, the search time increases accordingly. In order to reduce the computational cost at the search time, we store the positions, velocities, and orientations of n target joints in a kd-tree. At run-time, our system searches for k-nearest motion examples (k = 8 in our experiments) efficiently using the kd-tree [53].

Training data offer both input and output data for the construction of the online local model. From the collected poses, we extract the positions and orientations of n target joints, which are then used as the input of the model. The input vector $\mathbf{x} \in \mathbb{R}^{7n}$ consists of the xyz position and quaternion of the joints. All joint quaternions of the character $\mathbf{y} \in \mathbb{R}^{76}$ are also extracted so that they can be used as the output of the model. As a training model, we employ kernel

CCA-based regression method, which is described in detail in the following section.

3.4.2 Kernel CCA-based Regression

Producing faithful character animation is a mathematically ill-posed problem because the input data from the sensors are not sufficient to determine the full degrees of freedom of a character. We choose a data-driven nonlinear statistical approach, kernel CCA-based regression, to find a mapping function between the low-dimensional input data and a high-dimensional character pose. Compared to other linear models, kernel CCA is capable of extrapolation. Our system can faithfully generate reasonable animation results when few similar motions exist.

CCA is a machine learning algorithm that maximizes the correlations between training inputs and outputs by extracting the meaningful features of both training inputs and outputs. Once the correlation has been maximized with the transformed training data, it becomes possible to efficiently avoid a large amount of error, which is typically associated with the regression process. We formulate this into a nonlinear problem while utilizing the kernel method in CCA [52], as there is a very clear nonlinear relationship between the training input **x** and output **y**. The following equation represents the nonlinear transformation of the training input x to a vector $\phi(\mathbf{x})$ in the higher-dimensional feature space:

$$\phi : \mathbf{x} = (x_1, \ldots, x_n) \rightarrow \phi(\mathbf{x}) = (\phi_1(\mathbf{x}), \ldots, \phi_N(\mathbf{x})) \quad (N > n)$$

Given m pairs of training data, $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m)$ and $\mathbf{Y} = (\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_m)$, the equation

of kernel CCA with the correlation coefficient ρ is defined as follows:

$$\max_{(\hat{\mathbf{x}},\hat{\mathbf{y}})} \rho = \max_{(\hat{\mathbf{x}},\hat{\mathbf{y}})} \frac{\hat{\mathbf{x}}^T \mathbf{H}\mathbf{F}\hat{\mathbf{y}}}{\sqrt{\hat{\mathbf{x}}^T \mathbf{H}^2 \hat{\mathbf{x}} \hat{\mathbf{y}}^T \mathbf{F}^2 \hat{\mathbf{y}}}} \tag{3.3}$$

where $\mathbf{H} = \phi^T(\mathbf{X})\phi(\mathbf{X})$ and $\mathbf{F} = \phi^T(\mathbf{Y})\phi(\mathbf{Y})$ represent the nonlinear version of the training data in matrix form. The derivation of Equation (3) is described in [52]. The i-th basis of $\phi(\mathbf{X})$ and $\phi(\mathbf{Y})$ can be represented as $\phi(\mathbf{X})\hat{\mathbf{x}}_i$ and $\phi(\mathbf{Y})\hat{\mathbf{y}}_i$, respectively, where $\hat{\mathbf{x}}_i$ and $\hat{\mathbf{y}}_i$ are the i-th coefficient vectors with respect to the training data pairs [52]. From the total m basis sets, we select l basis sets to reduce the dimension of the input training data. The extracted sets, $\{\hat{\mathbf{x}}, \hat{\mathbf{y}}\}$ can be represented as matrices, $\hat{\mathbf{X}} = (\hat{\mathbf{x}}_1, \hat{\mathbf{x}}_2, ..., \hat{\mathbf{x}}_l)$ and $\hat{\mathbf{Y}} = (\hat{\mathbf{y}}_1, \hat{\mathbf{y}}_2, ..., \hat{\mathbf{y}}_l)$.

Although it is difficult to determine the transformation function ϕ explicitly, we can solve this problem using an approach known as the kernel trick. The kernel trick uses a kernel function with well-defined inner products of vectors in the feature space. The kernel trick method can be applied to kernel CCA (Equation (3)) as the equation contains the values of the kernel for every instance of the training data [52]. We utilize the Gaussian kernel function, $k(\mathbf{x}, \mathbf{x}_j) = \exp\left(\frac{-\|\mathbf{x}-\mathbf{x}_j\|^2}{2\sigma^2}\right)$. The standard deviation of the Gaussian kernel is automatically determined by the standard deviation of the training input data. The reduced training input data can be computed as follows: $\bar{\mathbf{x}} = \hat{\mathbf{X}}^T \tilde{\mathbf{x}}$, where $\tilde{\mathbf{x}} = \begin{pmatrix} k(\mathbf{x}, \mathbf{x}_1), \ldots, k(\mathbf{x}, \mathbf{x}_m) \end{pmatrix}^T$. The training output data in the reduced dimension $\bar{\mathbf{y}}$ can be computed similarly.

We can obtain the matrix $\mathbf{A}$, which transforms the reduced input to the reduced output with the equation below:

$$\min_{\mathbf{A}} \sum_{i=1}^{m} \|\mathbf{A}\bar{\mathbf{x}}_i - \bar{\mathbf{y}}_i\|^2 \tag{3.4}$$

We can also obtain the transformation matrix $\mathbf{B}$, which transforms the reduced output to

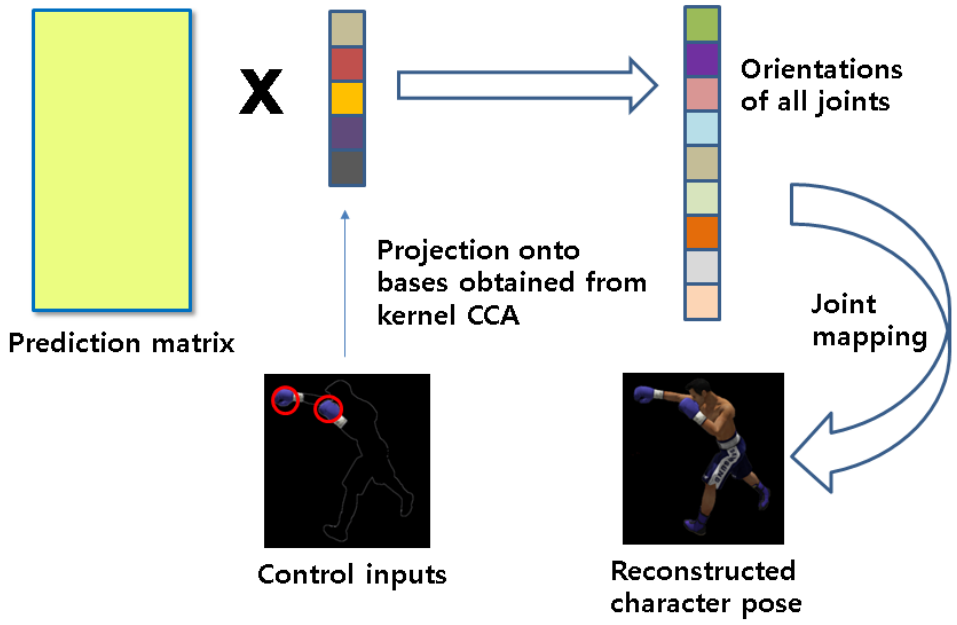

Figure 3.3: *An illustration of our kernel CCA-based regression model for online motion synthesis.*

the character pose $\mathbf{y}$ with the following equation:

$$\min_{\mathbf{B}} \sum_{i=1}^{m} \|\mathbf{B}\bar{\mathbf{y}}_i - \mathbf{y}_i\|^2 \tag{3.5}$$

We use the efficient LU solver [13] to process these two regressions. The desired pose is estimated by multiplying $\mathbf{BA}\hat{\mathbf{X}}^T$ by the kernelized input vector $\tilde{\mathbf{x}}$ (see Figure 3.3).

3.4.3 Motion Post-processing

While the character poses produced in the previous section are plausible in each frame, they may be temporally inconsistent. Because data from motion sensors are prone to temporal jitter, a smoothing process needs to be applied in order to achieve high-quality animation.

To generate a smooth animation result, two smoothing methods are jointly applied. In the first method, more input training data for the prediction model are used. We additionally use the nearest poses of the previous frame on top of those of the current frame to build the regression model. Second, we blend the resulting character poses of previous frames with the current character pose over all joints. The interpolating function is defined as follows:

$$\mathbf{y} = \mathbf{y}_t \otimes ((\mathbf{y}_{t-1} \otimes (\mathbf{y}_{t-2} \oslash \mathbf{y}_{t-1})^{\beta}) \oslash \mathbf{y}_t)^{\alpha}, \tag{3.6}$$

where $\mathbf{y}_t$ is the reconstructed pose in the current frame. Here, $\mathbf{y}_{t-1}$ and $\mathbf{y}_{t-2}$ denote the reproduced poses in the previous two frames. We use the mathematical notations introduced in [39, 43]. In our experiments, the blending weights were $\alpha = 0.3$ and $\beta = 0.4$. Finally, analytical inverse kinematics is applied to the character so that its end-effectors will accurately follow the position and orientation of the 3D motion sensors.

3.5 Experimental Results

We present the performance animation results of various motions and evaluate our system with different sensor setups and query metrics, and finally compare our system with three previous performance animation approaches. Motion data examples of boxing (7,233 frames), table tennis (4,271 frames), and walking (1,317 frames) were employed in these experiments.

Figures 3.4 and 3.5 depict the performance animation results given the performer's motion of boxing and table tennis, respectively. Both motions are consistently reconstructed using our method. In the boxing example, various types of punches are generated by our method,

Figure 3.4: *The boxing motion of the performer is transferred to the virtual character.*

Figure 3.5: *Realistic table tennis motion is reproduced by our system.*

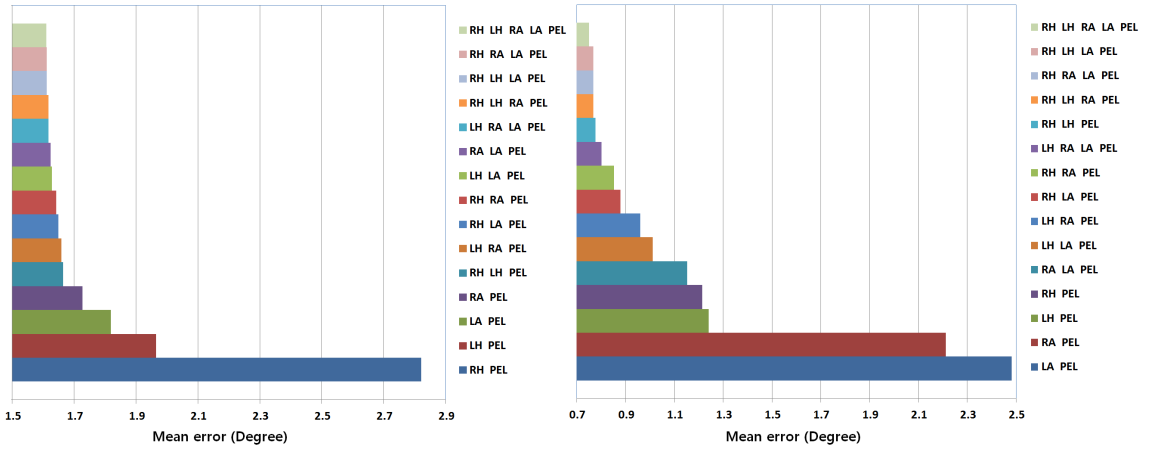

Figure 3.6: *Comparison of reconstruction error with different sensor setups: (left) walking motion; (right) boxing motion; RH: right hand, LH: left hand, LA: left ankle, RA: right ankle, PEL: pelvis.*

such as jabs, straights, hooks, and uppercuts. For the table tennis sequence, plausible motions such as forward/backward attacks and side-walking are successfully generated. Our system is capable of approximately 80 fps on a standard PC with an Intel Core i7 CPU and 16GB of memory. We include an accompanying video of the full-animation sequences.

Only one sensor on the right hand and five sensors on the target joints of the performer were used, and a side-by-side comparison was made. With only one sensor, the root position of the character automatically follows the destination of the ball without application of the inverse kinematics method to the feet. Our system generates a more natural motion sequence with five sensors than with only one sensor.

Regarding the walking and boxing motions, we observed the resulting errors according to variation of the sensor setups to select appropriate sensors. In this experiment, we use c_p = 0.6, c_q = 0.2, and c_v = 0.2 as the weight values for each respective term and set all other

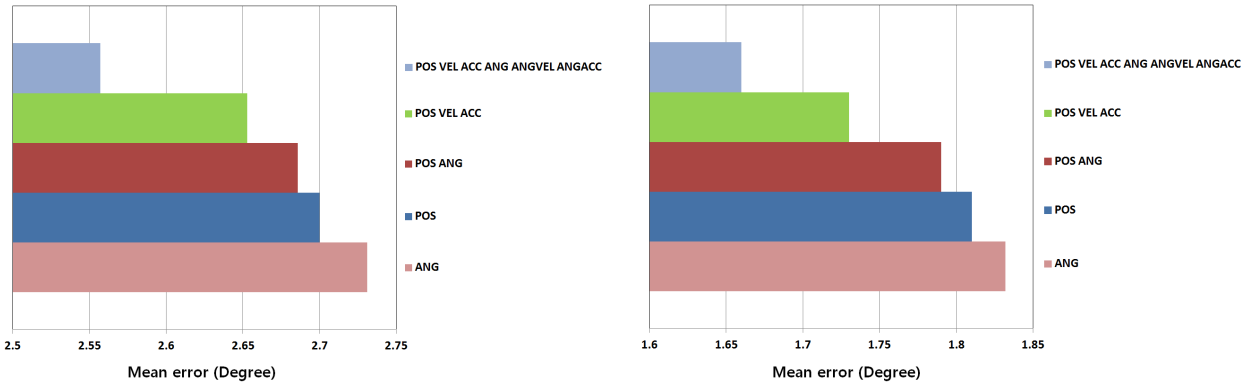

Figure 3.7: *Comparison of reconstruction error with combinations of features for the query metrics: (left) table tennis motion; (right) walking motion; POS: position; VEL: velocity; ANG: orientation; ANGVEL: angular velocity; ACC: acceleration; ANGACC: angular acceleration.*

weights to be zero. In Figure 3.6, the vertical axis of the graph represents all possible types of combinations while the horizontal axis shows the amount of error. We tested 15 possible sensor setup combinations. For walking motions, the placement of three sensors (RA, LA, and PEL) gave a suitable result compared to when more than three sensors were used. For the boxing motion data, with three sensors, their placement positions (RH, LH, and PEL) led to a well-generated result. In this experiment, we found that selecting the highest number of the most movable joints as the system input for each movement is important when seeking to minimize the number of sensors and reconstruct the desired motion sequences. As expected, more natural motion was generated when we added more sensors on the end-effectors on the limbs.

We evaluated the reconstruction error of different combinations of signals from 3D motion sensors to define the query metrics (Section 3.4.1). In Figure 3.7, the horizontal axis

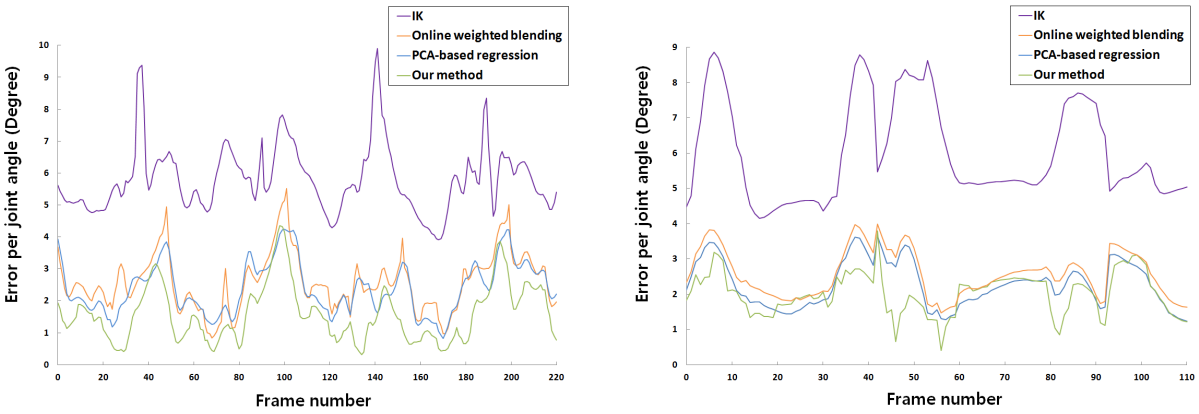

Figure 3.8: *Comparison of the joint angle error in the reconstructed motion with three baseline methods: (left) table tennis motion; (right) boxing motion; our method makes the smallest amount of error among all methods.*

shows the amount of error while the vertical axis represents the eight possible combinations of features from 3D motion sensors for the query metrics. These results show that the combination of position and orientation can generate more accurate results than with the independent use of position or orientation. For instance, in order to correctly identify the performer's forward swing during table tennis, we need both position and orientation features. Even in the same position, the type of swing may be different depending on the orientation of the hand. We also considered the velocity, angular velocity, acceleration, and angular acceleration as features for the query metrics to distinguish the direction of a given pose (e.g., forward and backward swings in table tennis). Based on these experiments, we exploit all 3D signals as query metrics to select the appropriate training poses. The weight of each term of the query metrics (Section 3.4.1) is determined to be proportional to the inverse of the total amount of error.

We compared the performance of our algorithm with three baseline methods: inverse kine-

matics, an online weighted blending method, and PCA-based regression. The inverse kinematics method simply matches all measured end-effector configurations to reproduce character poses. The online weighted blending method generates new poses on the fly via weighted blending of nearby k-neighbor poses (Section 3.4.1). PCA-based regression reproduces the motion sequences by utilizing two-step regression (Section 3.4.2). To test the extrapolation capacity of our method, we established a motion database with extremely sparse motion examples of total 85 frames (45 frames from table tennis motion and 40 frames from boxing motion), and reconstructed a target test pose that is not in the motion database. The graphs show that our method generates more accurate results than the other three methods. Figure 3.8 shows a frame-by-frame performance comparison of the given motion sequences.

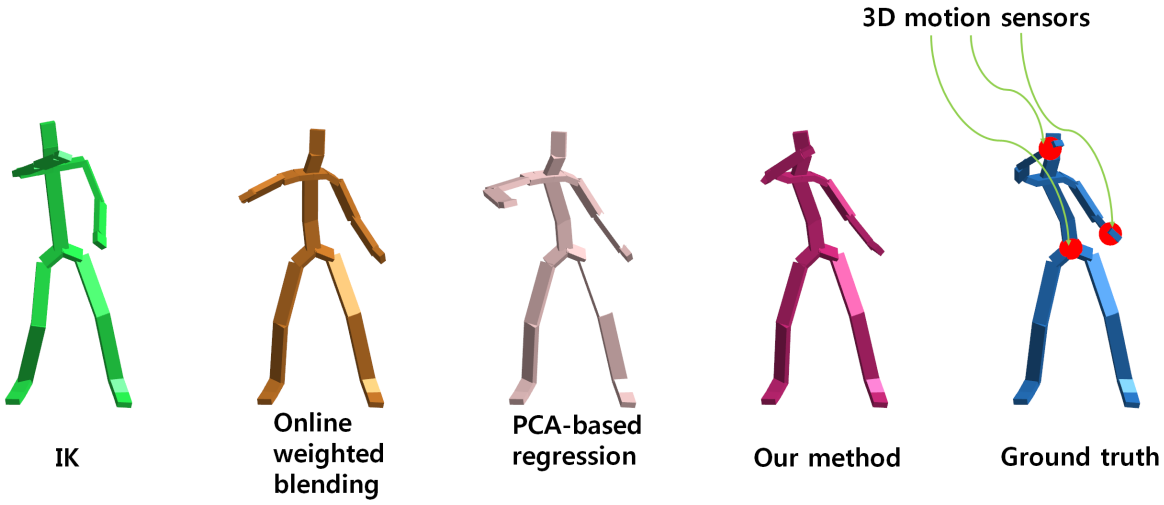

Figure 3.9: *Our method can generate the desired pose more accurately than the other methods due to the extrapolation capability of kernel CCA-based regression (See the right arm and spine). Three red spheres on the rightmost character indicate the locations of the 3D motion sensors (right hand, left hand, and pelvis).*

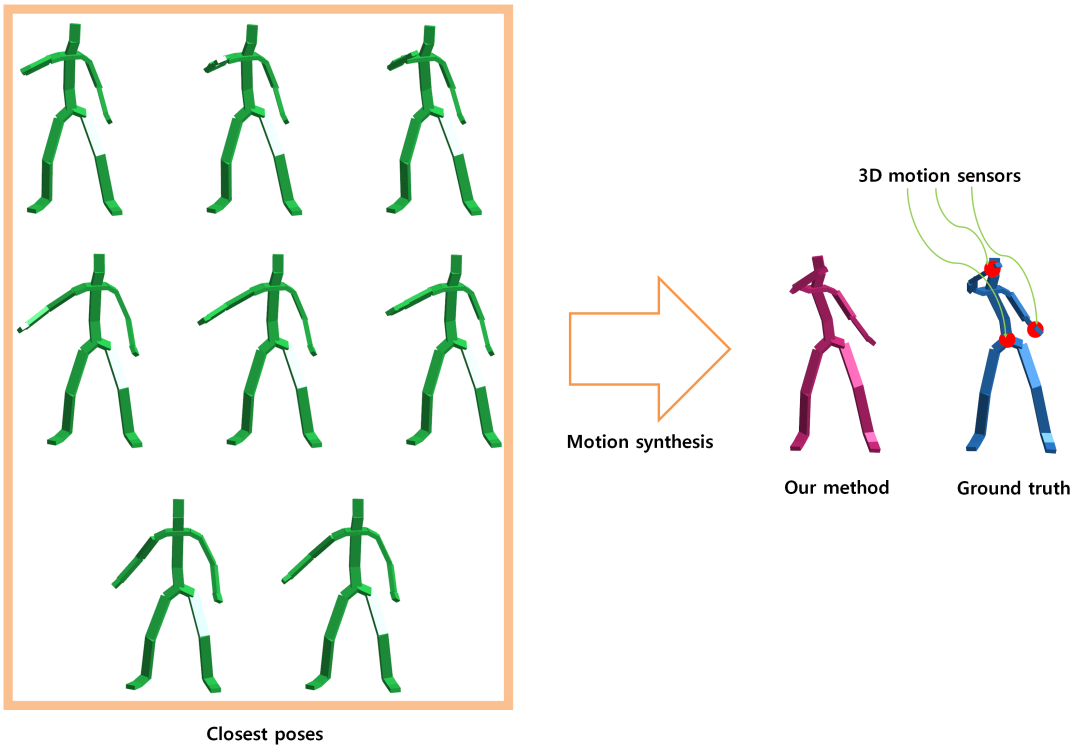

Figure 3.10: *Visualizations of the eight nearest neighbors for the corresponding synthesized pose and ground truth. Given the extrapolation capability of our approach, we can generate new poses that do not exist in the motion database.*

The extrapolation capacity of kernel CCA-based regression approach is evaluated by comparing our results with those from the baseline methods when the input motion sequences do not exist in the sparse dataset (table tennis, 40 frames of motion) as shown in Figure 3.9. All character poses are reconstructed without application of the inverse kinematics method to the end-effectors. In the event of insufficient motion capture data, our method can generate reliable results due to its extrapolation ability. Ground truth data are acquired from the motion capture data. Figure 3.10 shows the closest neighbor poses and the final synthesized pose.

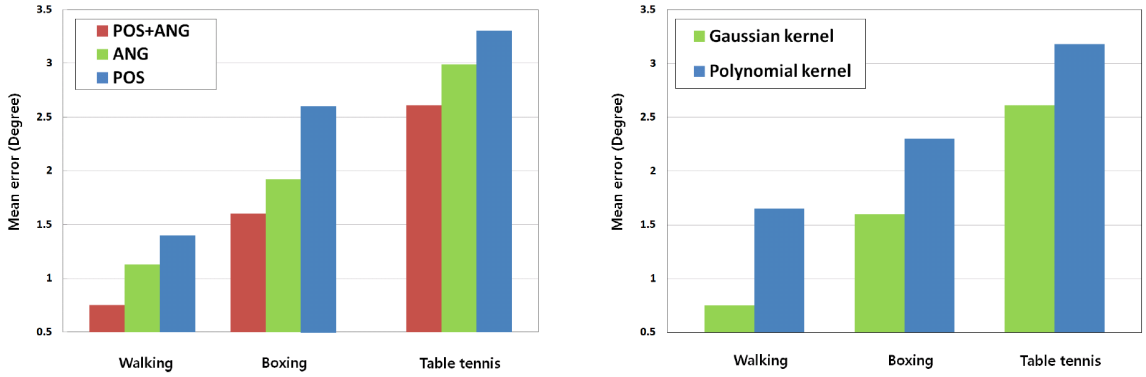

Figure 3.11: *Comparison of reconstruction error for three different actions: (left) motion reconstruction error with different combinations of sensor signals; (right) motion reconstruction error with different types of kernel functions.*

In kernel CCA stage, we experimented with another type of kernel function. We chose the polynomial kernel, $k(\mathbf{x},\mathbf{y}) = (\mathbf{x}^T\mathbf{y}+c)^p$, where p represents the degree of the polynomial and c is a constant value. The difference between the Gaussian and polynomial kernel is not significant in the predicted results. In most cases, the Gaussian kernel works well in all examples while the polynomial kernel produces slightly worse results when p is smaller than five. Figure 3.11 describes a comparison of the reconstruction error that is obtained with the use of different types of kernels.

The reconstruction error was compared with different combinations of input signals for learning the prediction model: position, orientation, and a combination of the two. The combination of position and orientation generated the most accurate results among them, because increasing the number of independent features typically leads to better performance, up to a point where the dimension of the input does not suffer the "curse of dimensionality" [14]. Figure 3.11 represents the performance comparison for different combina-

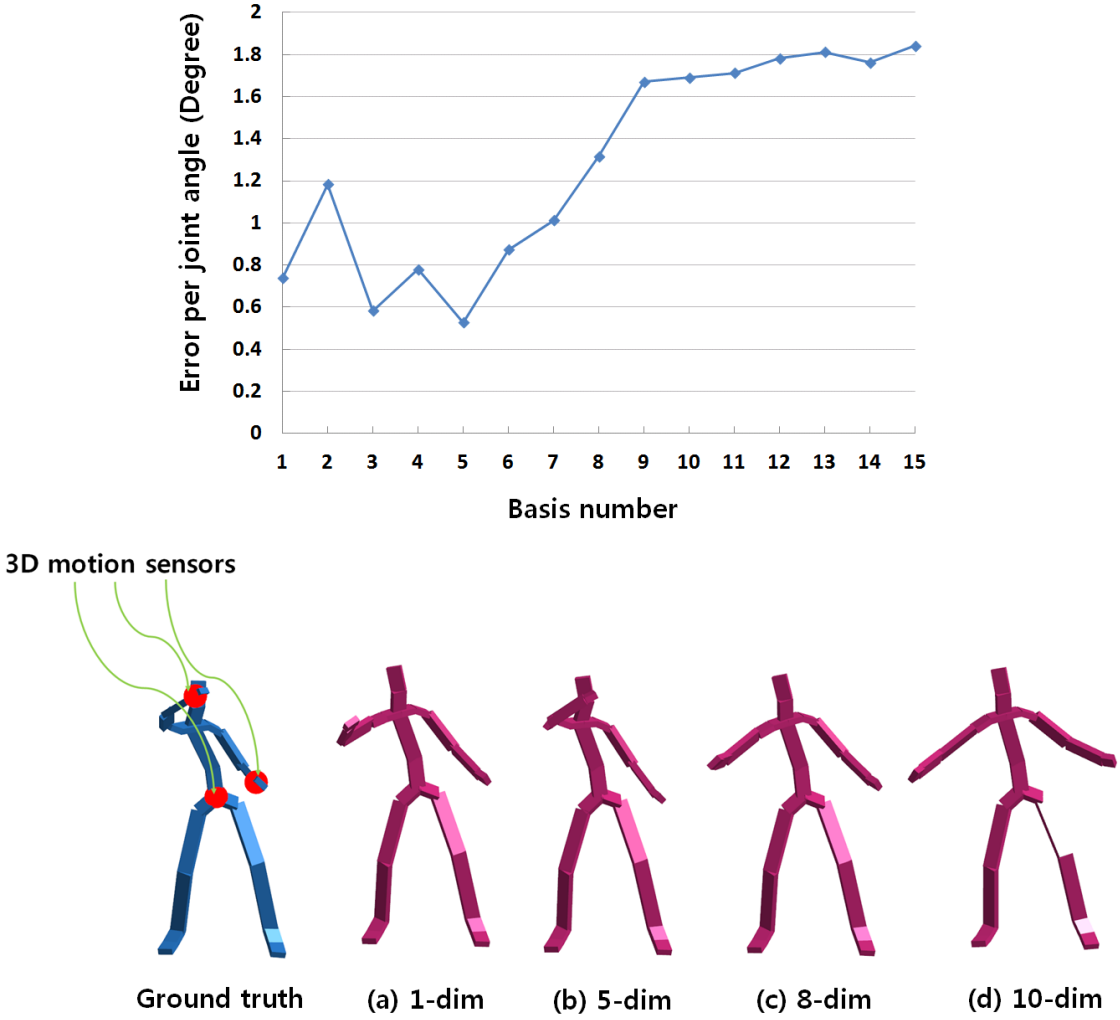

Figure 3.12: *Motion reconstruction accuracy for different basis number. The pose prediction error does not decrease monotonically as the basis number increases. Our system showed an over-fitting pose only when more than five dimensions were used. The reconstructed poses with different numbers of bases are illustrated along with the ground truth pose. In contrast to measuring the reconstruction error, we do not apply inverse kinematics to each illustrated character's end-effectors. dim: dimension.*

	length	**#neighbors**	**query time**
(a)	136s	8	2ms
(b)	225s	8	3ms
(c)	389s	8	4ms
(d)	389s	16	7ms

Table 3.1: *Length of motion database versus query time.*

tions of sensor signals.

We evaluated the motion reconstruction performance with respect to different numbers of kernel CCA bases. Given a large number of bases for kernel CCA, our model generated an over-fitting pose with more reconstruction errors. The most accurate pose was achieved with five bases. Figure 3.12 shows the reconstruction error for different number of bases.

The time complexity of our k-nearest search algorithm in d dimensions with n points and error bound e can be represented as $\mathcal{O}((c_{d,e} + kd)\log n)$ in [2], where $c_{d,e}$ is a constant term that is related to d and e. Our search algorithm efficiently finds training examples in large databases. The average query time (from 20 trials) was fast enough for real-time performance (Table 3.1). Theoretically, our system can handle up to a million of pre-recorded human poses.

3.6 Discussion

We have presented a robust real-time performance animation system based on kernel CCA-based regression framework. Kernel CCA suitably explains the nonlinear relationship between 3D motion sensors and character animation while generating more accurate char-

acter animation than previous statistical linear models. The improved performance of our approach was shown through an experimental evaluation.

To generate various types of motion sequences, we employ an online local model instead of a global one. After the online local model is established, our system generates new poses by means of simple matrix-vector multiplications; the matrix represents a linear transformation computed in a regression process and the vector represents the input data from 3D motion sensors. In addition, combining the nearby poses of the input motions of the current and previous frames' with effective temporal interpolation produces temporally smooth results. Temporal filtering works in real-time with a small delay. We demonstrated the robustness of our system by showing performance animation examples of boxing, walking, and table tennis.

To achieve more accurate results in the post-processing stage, a nonlinear programming approach with several equality constraints would be required. However, the fully nonlinear method runs too slow for real-time application. Given these limitations, our system provides a good balance between performance and precision/generalization, thus making it practical for various situations.

Similar to most sensor-based performance animation systems, our system may miss the position of 3D motion sensors when a performer moves rapidly. Although we can successfully approximate short-term missed sensors using a temporal smoothing function in the post-processing step, markers that are not tracked for a long time cannot be approximated. In this case, we should temporally stop the system and wait until reliable input data are provided.

While we concentrated on a sensor-based performance animation system in this study, our system can easily be extended by combining complementary types of input (e.g., Microsoft Kinect, the Vicon capture system, or stereo cameras). The use of multiple input sources would minimize the amount of missing data and improve the animation quality. We would need however to solve additional problems in order to achieve this goal, such as accurate synchronization and the appropriate selection of a model. This would be an interesting future research direction.

Chapter 4

Interactive Manipulation of Large-Scale Crowd Animation

Editing large-scale crowd animation is a daunting task due to the lack of an efficient manipulation method. We present a novel cage-based editing method for large-scale crowd animation. The cage encloses animated characters and supports convenient space/time manipulation methods that were unachievable with previous approaches. The proposed method is based on a combination of cage-based deformation and as-rigid-as-possible deformation with a set of constraints integrated into the system to produce desired results. Our system allows animators to edit existing crowd animations intuitively with real-time performance while maintaining complex interactions between individual characters. Our examples demonstrate how our cage-based user interfaces mitigate the time and effort for the user to manipulate large crowd animation.

4.1 Overview

Recent feature films and video games often show spectacular scenes with a large crowd of characters. Crowd animation may be used to show a variety of interesting events. The characters are sometimes crammed tightly into a small space. They sometimes interact with each other collaboratively or adversarially. There is a large array of crowd simulation algorithms that can generate a crowd of animated characters. These algorithms often have a set of tunable parameters. An animator adjusts the parameters to control how the crowd animation should look. The production of crowd simulation is a trial-and-error process of running multiple simulations with different parameters.

Alternatively, interactive editing techniques [37, 33] have been studied to allow animators to manipulate crowd animation directly. Their interactive techniques can effectively edit the group motion of multiple characters by pinning and dragging the characters at arbitrary timing while maintaining the integrity and visual quality of the original animation. The use of interactive editing techniques changes the animator's work process. The animator may begin with a simulation algorithm to generate an initial crowd animation from scratch and then he or she may manipulate the animation to refine the result interactively. Even though the group motion editing techniques have demonstrated promising functionalities that facilitate crowd simulation, their practical use has been limited because the techniques are computationally demanding. The previous work reported performance adequate for interactive dragging with dozen characters, which is far smaller than the large-scale crowds that often appear in feature films and games.

We present a novel cage-based editing method for manipulating large-scale crowd anima-

tion interactively. The cage is an abstract object that encloses target characters in motion for a certain spatial extent or temporal duration. The benefits of the cage-based approach are manifold. From the user interface point of view, the cage provides coherent control over multiple character motions such that the motions are manipulated in a coordinated manner. The cage also allows for the locality of control. The animator can manipulate a small portion of the animation enclosed by a cage, while leaving the remaining portions intact. The use of cages does not impose any limitation on human motion. Arbitrary types of human motion can be included in the crowd animation, even including interpersonal interactions with physical contacts. From the performance point of view, the cage serves as a reduced model of the animation; thus, it can be manipulated efficiently based on two-level hierarchical computation. Our method is faster than the previous Laplacian-based methods by orders of magnitude. The performance gain is even more substantial for a larger number of characters in a crowd.

Technically, our method is formulated as a quadratic programming with equality and inequality constraints. Interpersonal relationships, such as synchronization and physical contacts between characters, pose the equality constraints. We add the inequality constraints to prevent excessive deformation. The proposed method is inspired by a variety of previous work and consists of three major components. First, an as-rigid-as-possible mesh deformation model provides an effective means of manipulating cages. Second, the use of generalized barycentric coordinates establishes the correspondence between abstract cages and raw motion data. Finally, the quadratic programming with quadratic inequality constraints with both upper and lower bounds is intrinsically non-convex. We relax the non-convexity and solve it more efficiently. These components reinforce one another to achieve a scal-

able solution method. We will demonstrate the effectiveness of our method using examples including a dense crowd of characters interacting with each other.

4.2 Crowd Model

The *crowd animation* is a collection of motion clips that covers the extent of the animation. The full-body motion of an animated character is represented by its time-varying position and the orientation of the root segment in the global reference coordinate system together with the joint orientation of the remaining body segments. The motion may be captured, simulated, or procedurally generated. How the motion data is acquired does not matter. The *motion path* of a motion clip is a two-dimensional trajectory of the root segment projected onto the ground surface. Each motion clip is usually sampled uniformly and parameterized in local time. Situating a motion clip in a crowd animation maps the local time of the motion clip to the *time track* of a character. More specifically, let $\mathbf{P}^{\alpha}, \mathbf{P}^{\beta}, ..., \mathbf{P}^{N}$ be motion paths of N characters. Each motion path is represented as a piecewise linear curve, where the points on the curve are the time series of a character's location $\mathbf{p}_i^{\alpha} \in \mathbb{R}^2$ (see Figure 4.1). The direction of each point on a motion path is represented with respect to a local coordinate system defined by the tangent and normal vectors of the curve. The tangent and normal vectors at the i-th point are estimated by $(\mathbf{p}_{i+1}^{\alpha} - \mathbf{p}_{i-1}^{\alpha})$ and $R(\mathbf{p}_{i+1}^{\alpha} - \mathbf{p}_{i-1}^{\alpha})$, respectively, where R is a 2×2 rotation matrix of angle 90°.

The time track provides an efficient means of editing the relative timing of interacting characters. A collection of time tracks $\mathbf{T}^{\alpha}, \mathbf{T}^{\beta}, ..., \mathbf{T}^{N}$ of a crowd animation are vertically aligned to show the temporal relationships between characters. The timing of each motion

clip is locally parameterized by frame index i, which maps to $t_i^\alpha \in \mathbb{R}$ in synchronized reference time (see Figure 4.1). Manipulating the distances between the frame indices leads to time warping of the crowd animation.

The motion of one character at a certain time instance may be constrained with respect to the motion of another character. The interaction between characters and the environment induces spatial and temporal constraints. For example, consider two characters doing high five. Their relative location and timing of action should match at the moment when their hands meet. The animator may drag and pin down characters at certain time instances to introduce new constraints and thus manipulate the crowd animation. Editing crowd animation is a process of solving constraints between multiple character motions. The motion path and time track serve as spatial and temporal proxies of each motion clip. The constraint solving process results in the deformation of motion paths and time tracks. We then edit the full-body motions of characters accordingly to match their motion paths and time tracks.

Assuming that character α at frame i is interacting with character β at frame j, the spatial relationship between the characters is described as

$$\begin{aligned} \mathbf{p}_i^\alpha &= \mathbf{p}_j^\beta + c_x(\mathbf{p}_{j+1}^\beta - \mathbf{p}_{j-1}^\beta) + c_y R(\mathbf{p}_{j+1}^\beta - \mathbf{p}_{j-1}^\beta), \\ \mathbf{p}_j^\beta &= \mathbf{p}_i^\alpha + d_x(\mathbf{p}_{i+1}^\alpha - \mathbf{p}_{i-1}^\alpha) + d_y R(\mathbf{p}_{i+1}^\alpha - \mathbf{p}_{i-1}^\alpha), \end{aligned} \tag{4.1}$$

where (c_x, c_y) and (d_x, d_y) are the spatial coordinates of character α and β, respectively, with respect to each other. Their temporal relationship is specified such that

$$t_i^\alpha = t_j^\beta + g. \tag{4.2}$$

If $g = 0$, two character motions are exactly synchronized at frame i and j in reference time.

Symbol	Meaning
subscript i, j, k	frame index/ cage vertex index
subscript x, y	local coordinates in x, y axes
superscript α, β	character index
$\mathbf{p}_i^\alpha$	root position of character α at frame i
$t_i^\alpha, \mathbf{t}_i^\alpha$	reference time of character α at frame i, and the same symbol in cage-based editing
$\mathbf{v}_i, \mathbf{u}_i$	i-th boundary vertex on a spatial and temporal cage, respectively
λ_i^k, μ_i^k	MVC of k-th internal point w.r.t. $\mathbf{v}_i$ and $\mathbf{u}_i$, respectively
$\mathbf{c}, \mathbf{f}$	user specified constraints in space and time domain, respectively
$\mathbf{b}, \mathbf{g}$	relative interaction constraints in space and time domain, respectively

Table 4.1: *Mathematical notation.*

4.3 Cage-based Interface

Crowd animation includes a multiplicity of human behaviors happening concurrently. The interactive editing of crowd animation can be cumbersome if we have to manipulate individual characters independently. The user interfaces for crowd animation editing should provide effective means of exploiting the coherence and locality of the animation. Our cage-based interface serves this purpose (see Figure 4.1).

Global Coordination. The motion of multiple characters often needs to be manipulated in a globally coordinated manner in both space and time. The cage provides the grouping of characters such that the characters in the cage are manipulated coherently.

Locality. The user often wants to edit only a portion of an animation while the other part remaining intact. Our cage-based interface resembles the lasso tool in image editing software and supports various forms of locality, such as individuality and vicinity in space and

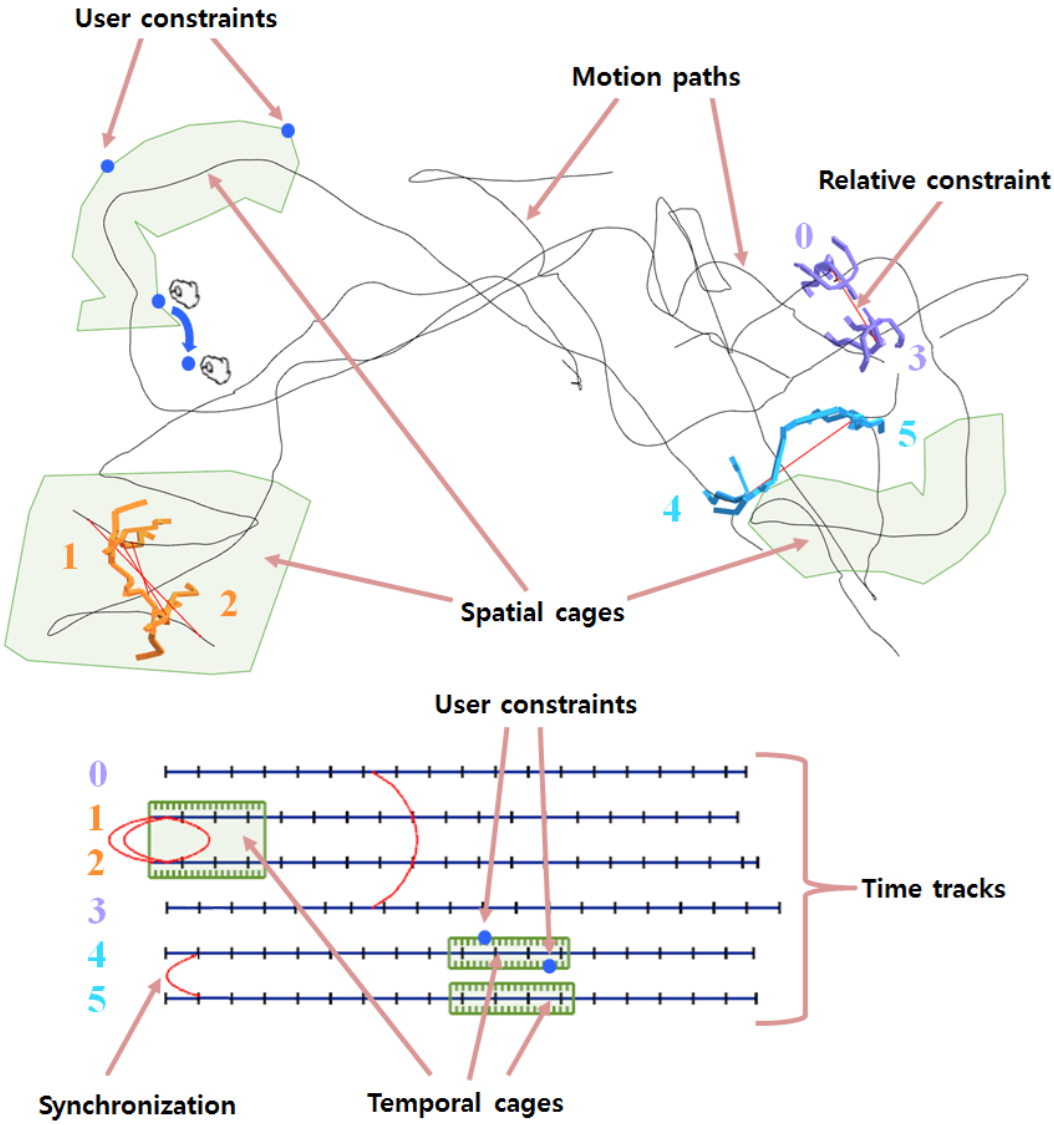

Figure 4.1: *Editing multi-character motions with cages. The motion paths and time tracks are enclosed by spatial and temporal local cages, which are shown in green. One of the spatial cages undergoes deformation given user constraints. The enclosed motion data are manipulated according to the cage deformation.*

time.

Versatility and Flexibility. The cage-based interface is versatile and flexible to deal with arbitrary types of human behaviors, which include locomotion, gestures, and even highly dynamic sport actions. Different types of human motions can be included in a single cage and manipulated simultaneously.

Interactive Performance. In general, a group motion is represented by using a large number of parameters that must be solved for every incident of user manipulation. The cage provides a compact representation of the group motion. The cage effectively encodes the course of action of multiple characters by using the generalized barycentric coordinates of sparse cage vertices. The use of reduced parameters achieves the interactive performance and scalability.

4.3.1 Cage Construction

The spatial cage is a polygonal mesh in two-dimensional space. The user may drag a free-hand selection around animated characters to select whole or portions of multiple motion paths. We construct the outer boundary of the cage using a concave hull algorithm [59], which computes a (possibly concave) polygon that encloses a set of input points with a certain margin. A large margin tends to make the boundary smooth, while a smaller margin makes the boundary highly concave. The offset distance from input points to the cage vertices is provided as a user parameter. Once the outer spatial boundary of the cage is determined, its interior region is tessellated by using constrained Delaunay triangulation [64]. The cage mesh is a jelly-like deformable object. The user can drag and pin down boundary vertices and/or arbitrary interior points to manipulate interactively. We will discuss cage

deformation later in Section 4.4. Our system also provides various editing operations to refine the selection. The cut operation allows the user to draw a cutting curve across any cage to divide it into two local cages. The user can also merge two cages, append a region to the selection, and crop to reduce the selection.

The crowd animation includes rows of time tracks. Each track corresponds to a course of action of an individual character. The temporal cage encloses whole or portions of multiple time tracks. All the operations defined for spatial cages can also be applied to temporal cages. In addition, the user is allowed to rearrange the order of characters interactively in the time tracks. The user can drag and drop rows, or sort the tracks by an order of character IDs, spatial locations (along x-axis, y-axis, or an arbitrary axis), or timing of action. A temporal cage selects a portion of crowd animation in the time domain and we can construct a spatial cage to enclose the same selected portion in the spatial domain. Similarly, we may draw a spatial cage first and then convert it into a temporal cage. The conversion between spatial and temporal cages is convenient if we want to select specific portions from a cluttered crowd scene. Figure 4.2 shows a conversion scenario. The motion paths are tangled in the spatial domain; thus free hand drawing may not allow us to precisely select desired portions, which are untangled in the temporal domain. The conversion capability allows us to select the portions in the time domain and then convert it into a compatible cage for spatial manipulation.

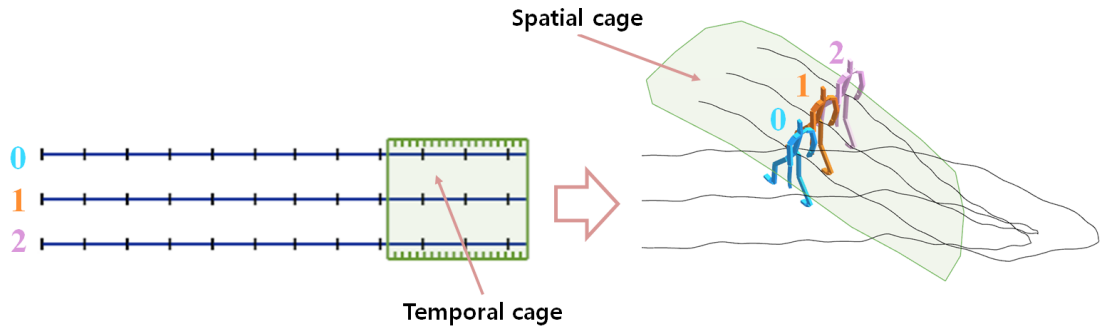

Figure 4.2: *Cage conversion from the time to the space domain.*

4.3.2 Cage Representation

The motion paths and time tracks inside the cage mesh are represented by MVC [16] with respect to the cage vertices. The reason we employ MVC is because of its smoothness. Tangent discontinuity is quite noticeable in character animation and should be avoided. Given point $\mathbf{p}_k$ on the motion path and cyclically-defined cage boundary vertices $\mathbf{v} = \{\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_m\}$, MVC $\{\lambda_1, \lambda_2, ..., \lambda_m\}$ of point $\mathbf{p}_k$ is defined by

$$\lambda_i = \frac{w_i}{\sum_{j=1}^{m} w_j}, \quad w_i = 2\frac{\tan(\phi_{i-1}/2) + \tan(\phi_i/2)}{||\mathbf{p}_k - \mathbf{v}_i||}, \tag{4.3}$$

where ϕ_i is angle $\angle \mathbf{v}_{i+1}, \mathbf{p}_k, \mathbf{v}_i$ in the polygon.

The interior of the cage is parameterized with MVC. Given the coordinates of a point, the location of the point is described as a weighted linear combination of the cage boundary vertices. When the cage vertex is updated, displaced point $\hat{\mathbf{p}}_k$ is computed as

$$\hat{\mathbf{p}}_k = \sum_{i=1}^{m} \lambda_i^k \hat{\mathbf{v}}_i, \tag{4.4}$$

where $\hat{\mathbf{v}} = \{\hat{\mathbf{v}}_1, \hat{\mathbf{v}}_2, ..., \hat{\mathbf{v}}_m\}$ are the updated boundary vertices.

MVC $\{\mu_1, \mu_2, ..., \mu_n\}$ of point $\mathbf{t}_k$ on the time track are defined in the same way with respect

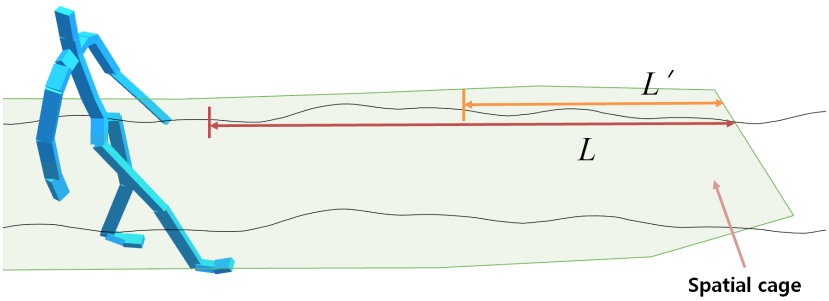

Figure 4.3: *Spatial local cage (green line) and its padding area.*

to temporal cage vertices $\mathbf{u} = \{\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_n\}$. In the temporal cage, the x-component of $\mathbf{u}_k$ denotes the reference time, and its y-component has a constant height value that corresponds to the order of a time track listed vertically. Any time instance on a time track can be described as a linear combination $\hat{\mathbf{t}}_k = \sum_{i=1}^{n} \mu_i^k \hat{\mathbf{u}}_i$ of the boundary vertices.

The local cage has a padding area along the boundary, which allows smooth transitioning of the interior parameterization to match the exterior of the cage. Let $\mathbf{p}_k$ be a point on the motion path, which will be displaced by the manipulation of the cage vertices. Let L be a user-specified padding depth, and let L' be the arc-length from $\mathbf{p}_k$ to the intersection of the motion path and the cage boundary (see Figure 4.3). Displaced point in the padding area is defined as $\hat{\mathbf{p}}_k = \gamma \sum_i \lambda_i \hat{\mathbf{v}}_i + (1-\gamma)\mathbf{p}_k$, where $0 \leq \gamma \leq 1$. Here, γ is an ease-in/ease-out function of arc-length ratio L'/L. We usually set L to the half of the arc-length of each individual motion path. This blending function is also useful when we cut any cage into two separate cages, which have a padding area along which motion paths blend in and out smoothly.

4.4 Editing Crowd Animation

The user interfaces for cage editing look similar to those of existing mesh editing techniques. The triangular or tetrahedral mesh can be regarded as a jelly-like deformable object, which changes its shape as little as possible under user manipulation. Unlike surface modeling, the cage of crowd animation encloses complex human actions and interpersonal interactions between multiple characters. This under-lying structure makes cage editing technically different from surface modeling.

We formulate the interactive manipulation of crowd animation as quadratic programming subject to several forms of constraints. The constraints arise from various sources, such as user manipulation, interpersonal interaction, character-environment interaction, collision avoidance, and excessive deformation of motion data. We classify constraints into three categories: linear equality, distance, and linear inequality. Each category requires different solution techniques. We will begin with a simple formulation including the objective function of spatial deformation and linear equality constraints. Then, we will discuss more complex formulations with the other forms of constraints.

4.4.1 Spatial Manipulation

Manipulation of a spatial cage is achieved by minimizing the objective function

$$E_D + w_c^2 E_C + w_m^2 E_M + w_b^2 E_B, \tag{4.5}$$

where E_D is the deformation energy, E_C denotes an energy arising from any errors with respect to the user constraints, E_M denotes the motion constraints induced from the in-

teractions between characters and the environment, and E_B is a term for preserving the relative distance between interacting characters. In our implementation, the weight values are $w_c = 100$, $w_m = 20$, and $w_b = 20$. We will discuss the first three terms and further elaborate the use of the fourth term later.

The deformation energy E_D measures the shape distortion of the cage mesh. We employ an as-rigid-as-possible deformation energy, which sums the local deformation energy over every triangle in the mesh to compute the global energy [68, 51]:

$$E_D = \sum_{t=1}^{T} \sum_{i=0}^{2} \cot(\theta_i^t) \| (\hat{\mathbf{v}}_i^t - \hat{\mathbf{v}}_{i+1}^t) - R_t(\mathbf{v}_i^t - \mathbf{v}_{i+1}^t) \|^2, \tag{4.6}$$

where $\hat{\mathbf{v}}_i^t$ is the vertex location in the deformed triangle. All subscripts denote modulo 3. Here, $\cot(\theta_i^t)$ is the per-edge weight for compensating non-uniform triangle shapes, θ_i^t is the angle opposite to the edge $(\mathbf{v}_i^t, \mathbf{v}_{i+1}^t)$ in the t-th triangle, and R_t is the estimated rotation matrix between the original and deformed shape of the t-th triangle.

The user can drag and pin down either the cage vertices or an arbitrary point inside the cage. Editing a specific motion path in a cage modifies the shape of a cage so that other motions in a cage can also be manipulated. Editing a cage vertex yields $\hat{\mathbf{v}}_k = \mathbf{c}_k$, where $\mathbf{c}_k \in \mathbb{R}^2$ is (x, y) position of the user constraint. The direct manipulation of an inside point yields $\sum_{i=1}^{m} \lambda_i^k \hat{\mathbf{v}}_i = \mathbf{c}_k$, where λ_i^k is MVC of the point. By concatenating all boundary vertices $\hat{\mathbf{v}} = \{\hat{\mathbf{v}}_i\}$ and user constraints $\mathbf{c} = \{\mathbf{c}_k\}$, we have another quadratic term $E_C = \|C\hat{\mathbf{v}} - \mathbf{c}\|^2$.

Motion constraints E_M preserves the relative location between characters, while they are interacting with each other. Replacing (x, y) coordinates with MVC in Equation (4.1), we have motion constraints in a quadratic form, $E_M = \|M\hat{\mathbf{v}}\|^2$.

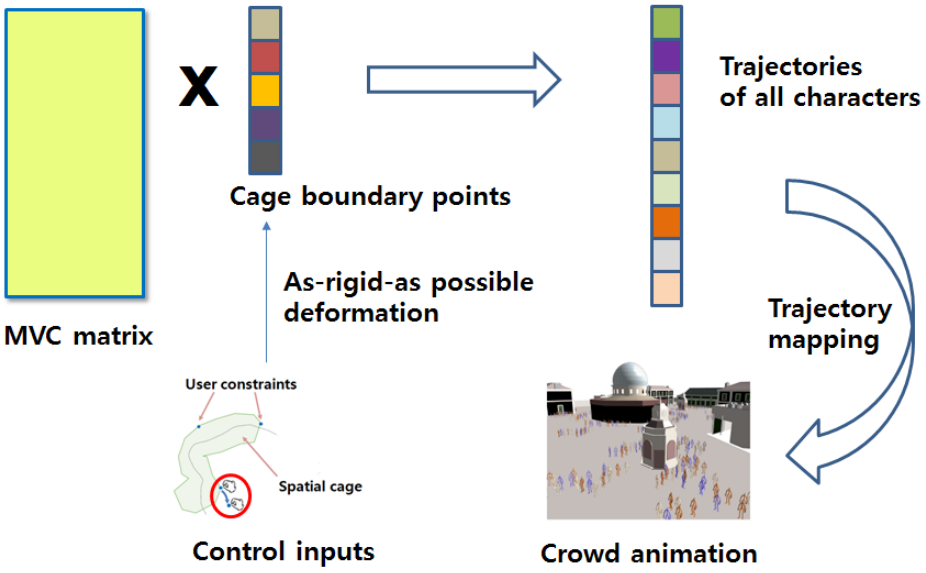

Figure 4.4: *An illustration of our cage-based editing model for manipulating crowd animation.*

Assuming that rotation matrices $\{R_t\}$ are fixed and taking the gradients of the quadratic energy terms, minimizing $E_D + w_c^2 E_C + w_m^2 E_M$ is reduced to a system of linear equations, which can be solved efficiently. Actually, the rotation matrices vary as the cage undergoes deformation. Solving the equation with unknowns $\hat{\mathbf{v}}$ and $\{R_t\}$ is a nonlinear optimization process. We employ an alternating least-squares optimization method [68]. Starting from the initial configuration $\hat{\mathbf{v}}$, rotation matrices $\{R_t\}$ are estimated by using singular value decomposition (SVD). We then solve for cage vertices $\hat{\mathbf{v}}$ while fixing $\{R_t\}$. These two steps iterate alternately. The optimization converges quickly within several iterations in most of our examples.

Distance Preservation. The fourth term E_B penalizes the distance changes between two interacting characters. The relative spatial constraints in Equation (4.1) are specified with

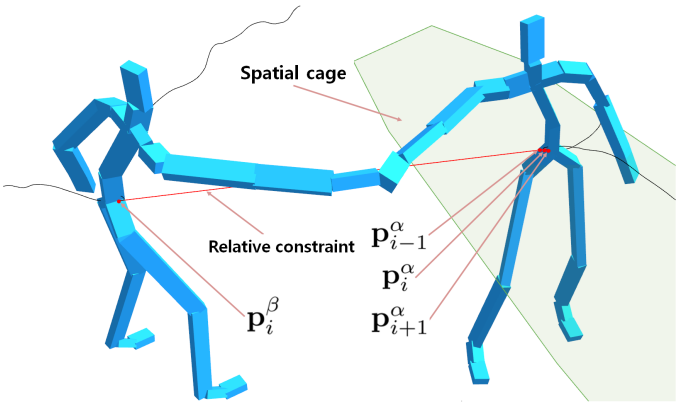

Figure 4.5: *A relative constraint is separated by the cage boundary.*

respect to the local coordinate system estimated from the motion path. Stretching of the motion path skews the local coordinate system; hence, the distance between two interacting characters scales accordingly. Note that E_M does not penalize the distance change induced from the skewing of the local coordinate system. E_B works in the iterative process. We run the optimization without E_B in the first iteration, and E_B becomes effective starting from the second iteration. Let $\hat{\mathbf{p}}_i$ and $\bar{\mathbf{p}}_i$ be motion paths at the current and previous iterations, respectively. The distance preserving constraint is formulated as

$$\hat{\mathbf{p}}_i^{\alpha} - \hat{\mathbf{p}}_j^{\beta} = \frac{\|\mathbf{p}_i^{\alpha} - \mathbf{p}_j^{\beta}\|}{\|\bar{\mathbf{p}}_i^{\alpha} - \bar{\mathbf{p}}_j^{\beta}\|}(\bar{\mathbf{p}}_i^{\alpha} - \bar{\mathbf{p}}_j^{\beta}). \tag{4.7}$$

Concatenating all distance-preserving constraints and plugging MVC into constraint equations, we have a quadratic energy term $E_B = \|B\hat{\mathbf{v}} - \mathbf{b}\|^2$.

Handling Local Cages. In the presence of a local cage, a relative constraint between two interacting characters may intersect with the cage boundary, making one character stay inside the local cage and the other outside the cage (see Figure 4.5). To maintain the original interaction between characters, we constrain the position and the tangent direction of the

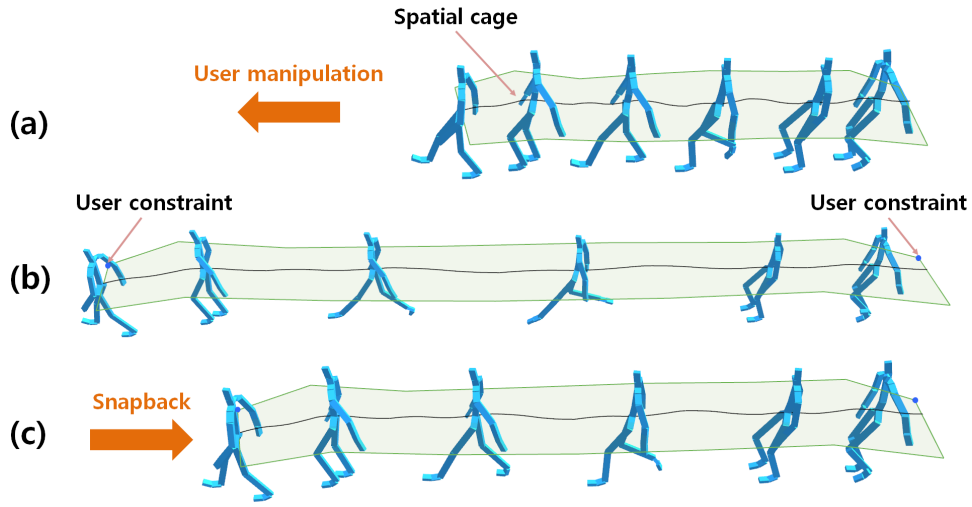

Figure 4.6: *Excessive manipulation on the original motion may result an unnatural motion ((a) original motion, (b) resulting motion). Our system automatically snaps the motion path back to the allowable range of motion when the user releases the motion path from dragging.*

motion path at the constrained point $\mathbf{p}_i^\alpha$ inside the cage. This can be simply achieved by making point $\mathbf{p}_i^\alpha$ and its two neighboring points $\mathbf{p}_{i-1}^\alpha$ and $\mathbf{p}_{i+1}^\alpha$ fixed during user manipulation.

Spatial Snapback. An excessive deformation on original motion paths may result in unnatural character motion, but it is not easy for the user to check individual character motion in a large crowd scene. We provide an automatic spatial snapback to constrain the motion in a reasonable range. Whenever the user releases from dragging, the system checks the deformed motion paths. If the deformed motions exceed the allowed range of editing, the system reduces the weight value of the active dragging handle and adds inequality conditions to snap the motion paths back into an allowed range (see Figure 4.6). We formulate

this problem as a quadratically constrained quadratic program (QCQP):

$$\underset{\hat{\mathbf{v}}}{\text{minimize}} \qquad \hat{\mathbf{v}}^T S_0 \hat{\mathbf{v}} + \mathbf{s}_0^T \hat{\mathbf{v}} + s_0 \tag{4.8a}$$

$$\text{subject to} \qquad l_b^i \leq \hat{\mathbf{v}}^T S_i \hat{\mathbf{v}} \leq u_b^i, \;\; \forall i > 0, \tag{4.8b}$$

where S_0 is a positive semi-definite matrix, $\mathbf{s}_0 \in \mathbb{R}^{2m}$, $s_0 \in \mathbb{R}$, and l_b^i and u_b^i represent lower and upper bound, respectively, for the i-th inequality constraint. Merging the four terms of Equation (4.5) into a single quadratic form, Equation (4.5) is equivalent to Equation (4.8a). The length of each edge should not change beyond a certain threshold. The inequality constraint for an edge between the i-th and j-th cage vertices is $\tau_{min} \leq \frac{\|\hat{\mathbf{v}}_i - \hat{\mathbf{v}}_j\|}{\|\mathbf{v}_i - \mathbf{v}_j\|} \leq \tau_{max}$. In our experiments, threshold values are $\tau_{min} = 0.5$ and $\tau_{max} = 2$. The derivation of constraints on the edges leads to quadratic inequality forms, and all quadratic matrices $\{S_i\}$ in Equation (4.8b) are positive semi-definite. The QCQP problem in Equation (4.8a) and (4.8b) is intrinsically non-convex. We relax the non-convexity by converting it into a semi-definite programming (SDP), which is a subfield of convex optimization:

$$\underset{\hat{\mathbf{v}}}{\text{minimize}} \qquad \mathbf{Tr}(\hat{V} S_0) + \mathbf{s}_0^T \hat{\mathbf{v}} + s_0 \tag{4.9a}$$

$$\text{subject to} \qquad l_b^i \leq \mathbf{Tr}(\hat{V} S_i) \leq u_b^i, \;\; \forall i > 0 \tag{4.9b}$$

$$\begin{pmatrix} \hat{V} & \hat{\mathbf{v}} \\ \hat{\mathbf{v}}^T & 1 \end{pmatrix} \succeq 0, \tag{4.9c}$$

where $\hat{V} = \hat{\mathbf{v}}\hat{\mathbf{v}}^T$. We would like to refer the readers to [6] for detailed derivation. We use a convex optimization MATLAB toolbox to solve the SDP problem [70].

In addition, the character's motion along the deformed path requires a final touch-up to avoid foot-sliding artifacts. We use inverse kinematics to enforce the foot constraints at the end of editing process.

4.4.2 Temporal Manipulation

Cage editing in the temporal context aims to preserve the initial tempo of motions given user and motion constraints. Temporal cage editing is much simpler than spatial cage editing because cage vertices move only along the horizontal direction. The problem is also formulated as a quadratic programming (QP):

$$E_{D'} + w_{c'}^2 E_{C'} + w_{m'}^2 E_{M'}. \tag{4.10}$$

The deformation $E_{D'}$ is derived from the temporal difference $\hat{u}_{i+1}^x - \hat{u}_i^x = h_i$, where $\hat{u}_i^x$ is the x-component of the i-th cage vertex, and h_i is the temporal difference at the initial configuration. The temporal difference relation leads to a linear system, $H\hat{\mathbf{u}} = \mathbf{h}$; thus, the corresponding quadratic energy is derived.

Here, $E_{C'}$ and $E_{M'}$ are the energy functions derived from the user constraints and relative interaction constraints in the temporal domain, respectively. The user constraints on the time track are defined in the same manner as the spatial constraints. The user may manipulate a cage boundary vertex or an arbitrary point on any time tracks in the cage. Dragging is permitted only in the horizontal direction. The manipulation of a boundary vertex yields $\hat{\mathbf{u}}_k = \mathbf{f}_k$ and the direct manipulation of a time track yields $\hat{\mathbf{t}}_k = \mathbf{f}_k = \sum_i^n \mu_i^k \hat{\mathbf{u}}_i$. Concatenating all user constraints in the time domain forms energy function $E_{C'} = \|F\hat{\mathbf{u}} - \mathbf{f}\|^2$. The synchronization constraints in Equation (4.2) can be either specified manually by the user or computed automatically by analyzing motion data [32]. Replacing the time index with MVC of the temporal cage, the energy function also leads to a quadratic form $E_{M'} = \|G\hat{\mathbf{u}} - \mathbf{g}\|^2$. Note that unlike spatial editing, we do not need the fourth term for scaling compensation because scaling artifacts do not happen in one-dimensional time domain.

Temporal Snapback. The time should increase monotonically even under time warping. Excessive editing could cause time flipping that makes time go backward. We address this problem by incorporating monotonicity constraints into the QP:

$$\underset{\hat{\mathbf{u}}}{\text{minimize}} \qquad \hat{\mathbf{u}}^T T_0 \hat{\mathbf{u}} + \mathbf{t}_0^T \hat{\mathbf{u}} + t_0 \tag{4.11a}$$

$$\text{subject to} \qquad \mathbf{l}_b \leq T_1 \hat{\mathbf{u}} \leq \mathbf{u}_b, \tag{4.11b}$$

where T_0 is a positive semi-definite matrix, $\mathbf{t}_0 \in \mathbb{R}^{2n}$, $t_0 \in \mathbb{R}$, and $\mathbf{l}_b$ and $\mathbf{u}_b$ represent lower and upper bounds, respectively. Three terms in Equation (4.10) can be merged into a single quadratic equation in Equation (4.11a). The inequality constraint for the i-th cage vertex is $h_{min} \leq \frac{\hat{u}_{i+1}^x - \hat{u}_i^x}{u_{i+1}^x - u_i^x} \leq h_{max}$, where the threshold values are $h_{min} = 0.3$ and $h_{max} = 3$. Concatenating all inequality constraints yields Equation (4.11b). Similar to spatial snapback, we reduce the weight value of the active dragging handle to make the time tracks snap back to an allowed editing range.

4.5 Collision Avoidance

Characters in crowd motion data are tightly packed together and interact with each other in close proximity. Manipulating such crowd motion by deforming its overall shape can easily cause collisions between characters or between character-environment that did not present in the original motion.

Resolving collisions between characters and the environment geometry is relatively easy. If a collision is detected, we add a new constraint to push the character out of the boundary of the environment geometry. We repeat this process until all collisions are resolved. The

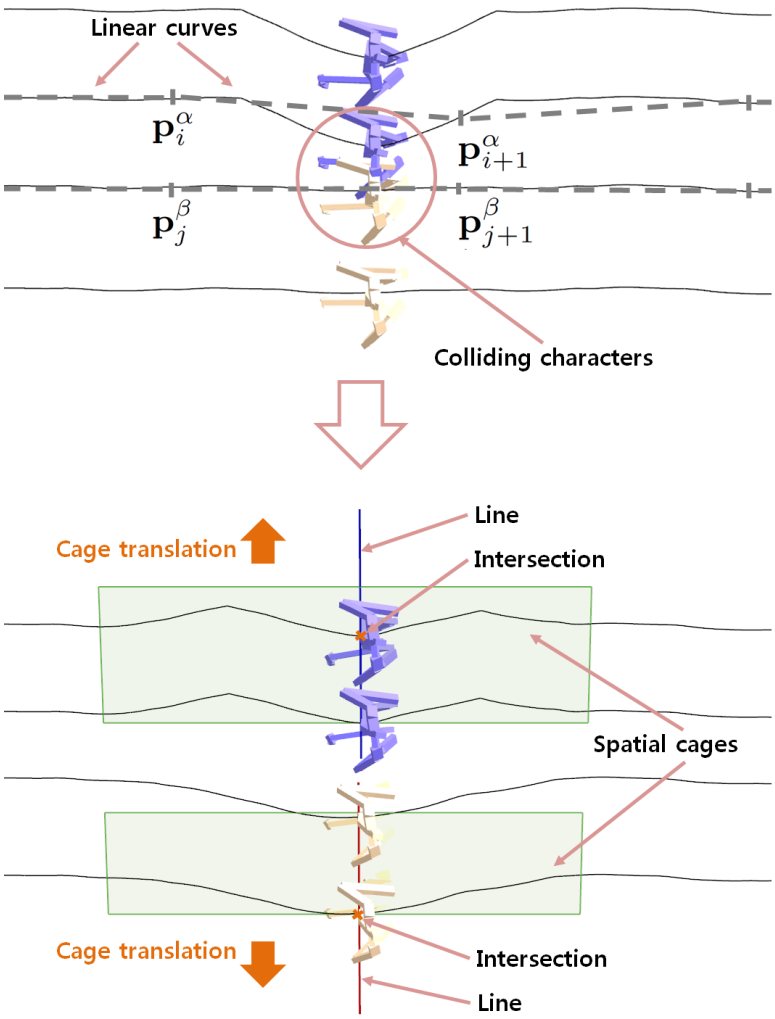

Figure 4.7: *Collsion handling. A collision between characters is detected using piecewise linear curves and neighboring characters for local cages are found by extending the line from the collision point. The collision is resolved via iterative local cage translation.*

iterative process is similar to the one suggested by Choi et al. [11].

A more challenging problem is to resolve collisions between characters. We provide a cage-based collision avoidance algorithm to address this problem. Specifically, we discuss efficient collision detection in large crowds, creating local cages by grouping characters

around collisions, and iterative relaxation of local cages for collision resolution.

Given a crowd animation, detecting collisions between characters at every frame is computationally overwhelming. We can approximate the original motion paths with sparse piecewise linear curves by subsampling path points at every second. Given two characters on the i-th and j-th linear curves, we determine a collision when the minimum distance between two curves is smaller than the user-defined threshold r. When this condition is satisfied, we find the exact colliding time t_{min} by minimizing $\|(\mathbf{p}_i^\alpha + t\mathbf{d}_i^\alpha) - (\mathbf{p}_j^\beta + t\mathbf{d}_j^\beta)\|$, where $\mathbf{d}_i^\alpha = \mathbf{p}_{i+1}^\alpha - \mathbf{p}_i^\alpha$ and $\mathbf{d}_j^\beta = \mathbf{p}_{j+1}^\beta - \mathbf{p}_j^\beta$. When $0 \leq t_{min} \leq 1$, we can easily identify the intersection point between two linear curves. If $t_{min} < 0$ or $t_{min} > 1$, we compute the per-vertex distance between two linear curves and find the exact time t_{min} where the distance is minimized (see Figure 4.7).

After finding t_{min}, we create two local cages that enclose each of the colliding characters and their coherent neighbors. Enclosing neighbors in the same cage is very important in order to prevent yet another collisions while fixing existing collisions. Two conditions must be satisfied for characters in a local cage: (1) the piecewise linear curve of a colliding character and its neighbors should be semi-parallel; (2) the line connecting two colliding characters must intersect the motion path of its neighbors (see Figure 4.7). We determine the size of the local cage by adjusting the length of the line. Large cages make smooth modification to motion paths, while small cages localize the effect of collision avoidance. In our experiments, the system pushes two local cages in opposite directions by the amount of $r/3$ at each iteration and repeats this process until all collisions are resolved.

Three simplification strategies are employed to boost the performance during the colli-

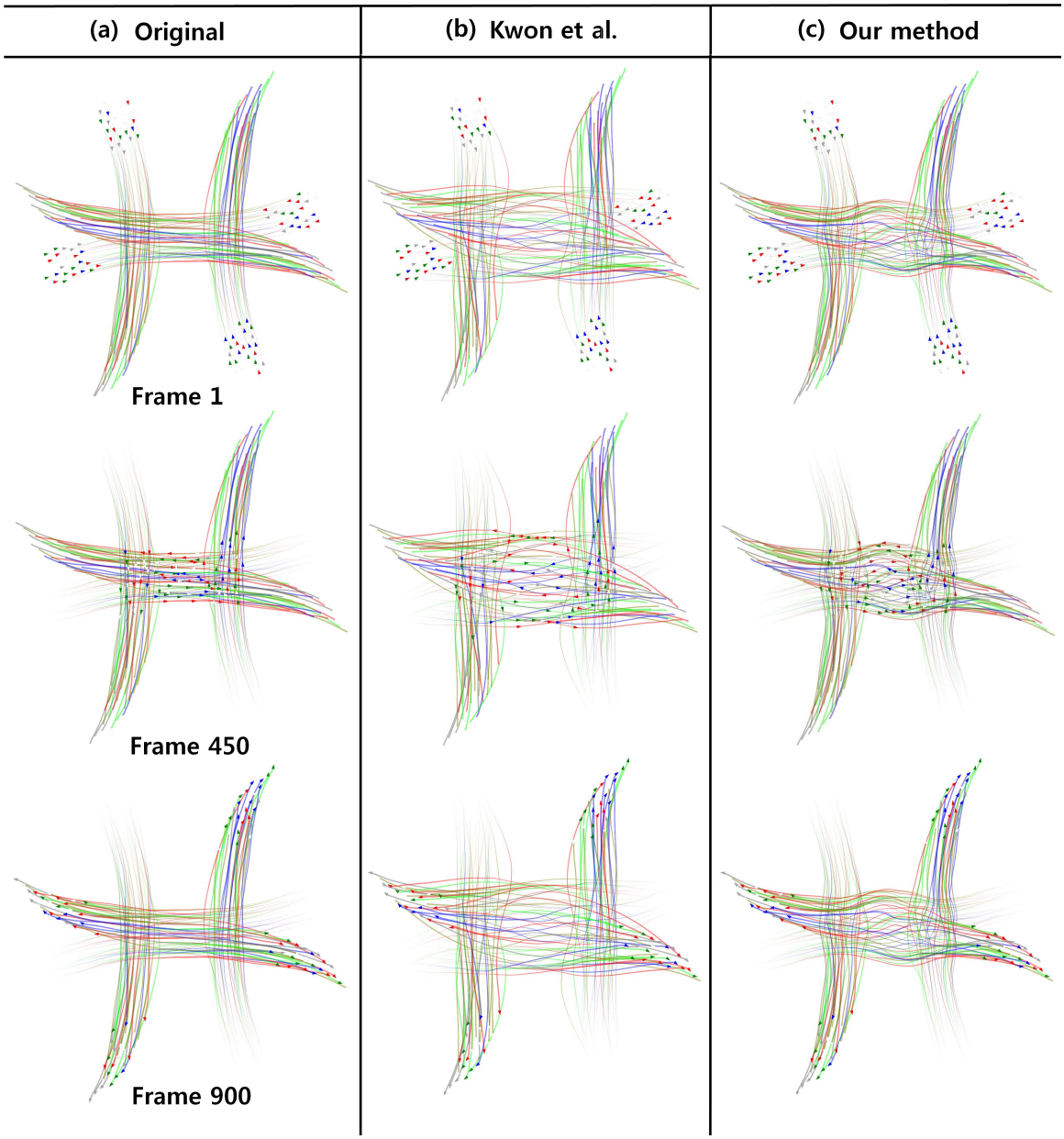

Figure 4.8: *Comparison of collision handling: (a) 100 characters from four directions crossing in a narrow space (b) collision avoidance by Kwon et al. [37] (240 sec.) (c) collision avoidance by our method (4 sec.); Our cage-based collision avoidance is much faster and causes less deformation.*

sion avoidance. First, each local cage has a rectangular shape with only four vertices to reduce the computation cost for MVC. Second, we use MVC only and one type of cage deformation, which is a rigid translation in opposite directions of the collision. Finally, we gradually reduce the size of local cages over the iterations to achieve fast convergence. With these strategies, we can obtain a collision-free crowd animation very efficiently. Figure 4.8 shows the collision handling results by Kwon et al. [37] and our method. Our method is about 60 times faster than the previous method using the same threshold r, and also shows less deviation from the original animation.

Collisions between characters need to be addressed when the user intentionally overlays a cage on top of another cage to make a complex crossing scene. When the cages are overlapped, our system uses the same collision detection method and the iterative local cage translation. After all collisions are resolved, the system allows further editing by providing either the original cages or a new single cage for all characters in the space. The accompanying video shows the collision avoidance example by such overlapping cages.

4.6 Experimental Results

The motion data used in our experiments were collected from various sources. Some data were acquired via motion capture, some were generated in crowd simulation, and some were synthesized using data-driven motion synthesis. Our examples show the process of motion editing from raw motion data. Each manual process took less than ten minutes. We refer the readers to the accompanying video clip to view the interactive process.

Small Town. In our first example, we begin with a collection of group motion clips (see Figure 4.9(a)). Each group motion clip was generated using a crowd simulator and a locomotion synthesizer. Individual clips include dozens of characters in a coherent group walking along a straight line, making turns, chatting in small groups, and hanging around. The editing process is to situate group motion clips into a new virtual environment by translating, rotating, deforming, stitching, cropping, dividing, copying, and pasting the clips. The cages were set up around individual group motion clips. The resulting crowd scene includes 900 characters and animation data of 521,600 frames in total. The editing process could have been extremely tedious if the animator had to manipulate the motion of each individual character independently. Our cage-based interfaces allowed us to create a desired crowd scene in only ten minutes, because we were able to manipulate one or two dozen characters simultaneously and coherently at each editing operation.

Racing Track. The second example used a single group motion clip that includes 60 characters running along a long straight line about 70 seconds. The environment is a racing track with circular edges and irregular obstacles. We used a single spatial cage enclosing the whole clip to adjust the raw motion data onto the racing track and then specified local cages where the characters penetrate through obstacles. Manipulating local cages to avoid obstacles is a straightforward process. There are low-hung ceilings in the environment. The characters have to slow down and crouch walk to pass below the ceilings. Slowing down the characters in the front row resulted in the interpenetration with the following characters. We used coordinated time warping to avoid collision between characters.

Chicken Hopping. "Chicken hopping" is a game often played by children. Players must

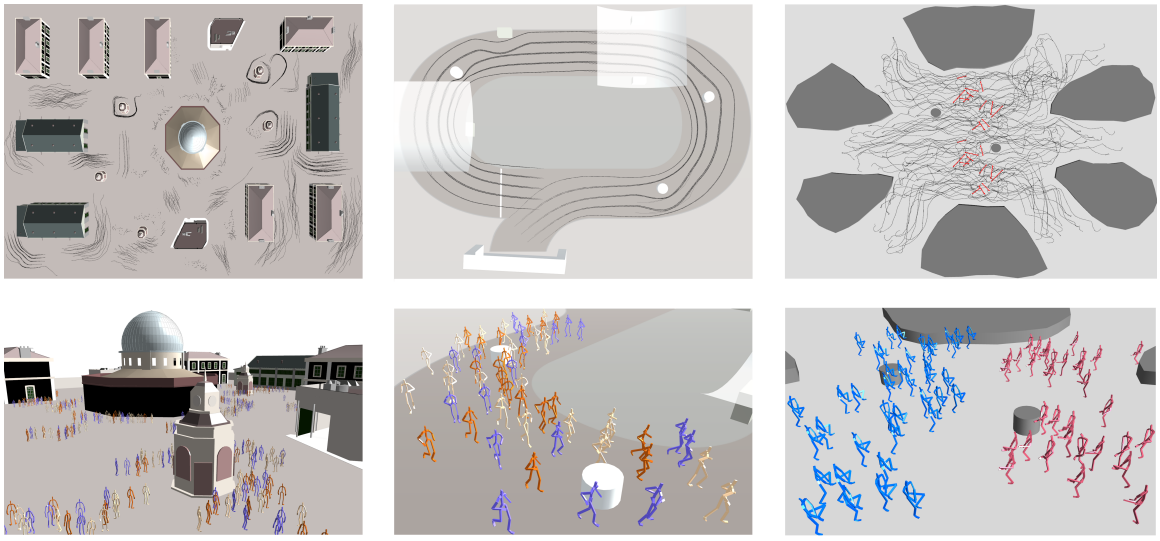

Figure 4.9: *Interactive editing of crowd animation: motion paths after editing (first row) and close-up view (second row); (a) small town, (b) racing track, and (c) chicken hopping.*

stand straight, lift up one of their legs, hold the ankle with their hands, and hop on the other leg. The players bump against each other to make their opponents lose balance. Kim et al. [32] generated an interesting crowd scene with 74 characters chicken hopping about 30 seconds. Manipulating such a data set is challenging because the character motions are densely packed in space and synchronized precisely in time. The user first draws a local cage on the beginning part of all time tracks. Cage conversion generates spatial cages at two corners of the environment, where two teams of players start to dash towards their opponents. The user then slices the local cages into smaller ones to make appropriate groups that support the editing requirement. The user also directly draws local cages in the space domain to edit small details. With our cage-based interfaces, we can easily manipulate the data to situate it in a new environment without interpenetration.

	# characters	# path points	# cage vertices	Solving	Multiplication	Snapback	Laplacian curve editing
(a)	78	14,040	15	1ms	1ms	176ms	392ms
(b)	118	21,240	15	1ms	2ms	176ms	635ms
(c)	398	71,640	30	2ms	9ms	387ms	2,231ms
(d)	600	230,600	40	3ms	42ms	621ms	7,324ms

Table 4.2: *Performance statistics. The performance of the solving and snapback steps depends only on the number of cage vertices, not on the size of the scene (see rows (a) and (b)). In the comparison with Laplacian curve editing, we used the same settings to compare the two methods (The snapback feature is not included in Laplacian curve editing). All performance in the table are the average computation time over 20 tests.*

Performance. Our system is fast enough to manipulate hundreds of characters at interactive frame rate (see Figure 1). The computational complexity of our algorithm is significantly lower than that of previous approaches [37, 33]. For a crowd animation with p characters and q frames, previous methods have to solve a series of $(pq \times pq)$ linear systems for every update, whereas our method solves $(m \times m)$ linear systems, where m is the number of cage vertices. In most cases, m is much smaller than pq, leading to faster update. Although our algorithm requires additional matrix-vector multiplication to update the motion paths and time tracks in the cages, this computation is as light as solving a linear system and can be effectively parallelized. We use OpenMP parallelization for computing the matrix-vector multiplication and achieved three times speed-up on an 8-core machine. Table 4.2 shows performance statistics. We stretched the cages to make the system perform the full operation of solving as-rigid-as possible mesh deformation, matrix-vector multiplication, and solving snapback. The last column of the table shows the performance of Laplacian curve editing by Kim et al. [33]. Note that Laplacian curve editing does not include snapback operation, so it should be compared to linear system solving and matrix-vector multiplication of our

algorithm. The performance data were measured on a standard desktop PC with an Intel i7 3.4GHz processor and 16GB of memory.

4.7 Discussion

We have presented an interactive method for editing a large crowd animation. Our cage-based approach has several advantages. Cage-based editing exploits spatial and temporal coherence between characters to deal with densely interacting characters in a coherent manner. Cage-based approach achieves significant performance gain over previous per-character methods. In practice, our method is almost two orders of magnitude faster than previous methods. We found that handling inequality constraints in quadratic programming is an essential component of the system if we want to manipulate a large, dense crowd of characters.

Table 4.3 summarizes the key aspects of our technique in comparison to previous crowd editing methods. Our method includes all positive aspects of the previous methods, and also provides additional advantages of global coordination, flexibility, locality, and performance. Given such advantages, we believe that our method can deal with a wide range of real world crowd editing tasks with its current form.

Existing crowd simulation methods and our interactive editing are complementary to each other. Even with the latest crowd simulation algorithm [58], it still remains difficult to precisely control individual character behaviors and details. Our algorithm can be used to make changes on top of the simulated results to refine it further.

	Kwon et al.	**Kim et al.**	**Our method**
(a)	motion path	motion path, time track	cage
(b)	locomotion	no restriction	no restriction
(c)	N	Y	Y
(d)	N	N	Y
(e)	moderate	slow	fast

Table 4.3: *Comparison with previous work. Five criteria are presented: (a) Targets for manipulation, (b) Type of motion, (c) interpersonal interaction, (d) locality, and (e) performance.*

Our approach has several limitations. First, MVC may have negative weight values for highly concave shapes and could occasionally result in unexpected cage deformation. Positive MVC [46] might alleviate this problem to provide non-negative weights even for highly concave shapes. Second, we currently do not allow for cages with open-holes. Open-holes can be a useful tool for avoiding collision between characters and obstacles. This problem can be addressed by using positive MVC with Delaunay smoothing [74]. Finally, our system only resolves interpenetration by warping spatial moving paths. Alternatively, it is also possible to resolve collisions by changing walking speed via time warping. We have not found a decision rule which strategy to choose between spatial and temporal warping. Designing a spatiotemporal warping strategy combining both would be an interesting direction for future research.

A crowd motion with many relative constraints has little room for further modification. For instance, in the chicken hopping example (Figure 4.9(c)), we could make only minor edits in the center of the scene and a lump of motion paths moved all together because many interactions occur frequently. It might be possible to remove and restore constraints incrementally to allow for further flexibility during interactive manipulation.

It is also possible to combine spatial and temporal manipulation into one 3D manipulation by using 3D spatiotemporal cages [31, 68, 4]. Manipulating a single cage can change both spatial paths and time tracks simultaneously. Our test implementation showed that the resulting interface could be confusing for users because it is not easy to predict the correlation between spatial and temporal manipulation. We think that the idea of 3D manipulation is promising, but it would require creative redesign of user interfaces to make it practically usable.

Chapter 5

Conclusion

This thesis presents a data-driven mapping model that transforms motion capture data to two different character animations system, performance animation and crowd animation. In the first animation system, we extract features from sensors and establish a realtime mapping model from motion capture data to synthesize a virtual character that reflects the user's actions as closely as possible. Various motions are reconstructed by our effective non-linear statistical method, kernel CCA-based regression. In addition, various combinations of features are evaluated and we select the most reasonable choice to obtain accurate results.

In the other application, we propose a manipulation mapping model to directly edit crowd animation so that it can complement several simulation based techniques. By using our application, an animator can easily manipulate some parts of the crowd simulation results to obtain more pleasing results. Our mapping model is fast enough to handle hundreds of

characters and preserve the spatial and temporal features of existing crowd animation.

Our mapping model can be extended to the medical field. Cerebral palsy is a disease causing physical disabilities and particularly impacts children. Stiff knee gait is a common disability associated with cerebral palsy. For example, the knee movement is significantly inactive, and maximum knee flexion in the leg swing is less than normal value. As a result, patients have many difficulties in their daily life. To improve the gait of patients, various types of surgical treatments are needed.

Even though rectus femoris transfer surgery can provide successful outcomes, it requires a long time to recognize whether or not the gait of post-operative patients has been mitigated. To address this problem, we can employ the proposed mapping model for immediately predicting the gait of post-operative patients because it can faithfully reproduce the gait of post-operative patients from the pre-captured motion data of patients before and after surgery.

We can also formulate a mapping model as a robust regression scheme with automatic feature selection. This mapping model automatically determines which combinations of features are best to reconstruct the gait of a post-operative patient. Meaningful features from patients' motion data provide useful information that can assist surgical direction of a clinical team. In this regard, we can consider a regression model based on sparse coding [38], which perform a regression to make a connection between the system input and output, and automatically finds sparse meaningful features by setting elements of the basis vectors that correspond to the irrelevant features to zero.

Bibliography

[1] Okan Arikan, David A. Forsyth, and James F. O'Brien. Motion synthesis from annotations. *ACM Trans. Graph.*, 22(3):402–408, 2003.

[2] Sunil Arya, David M Mount, Nathan S Netanyahu, Ruth Silverman, and Angela Wu. An optimal algorithm for approximate nearest neighbor searching. In *Proceedings of the fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 573–582, 1994.

[3] Norman I. Badler, Michael J. Hollick, and John P. Granieri. Real-time control of a virtual human using minimal sensors. *Teleoperators and Virtual Environments*, 2:82–86, 1993.

[4] Péter Borosán, Reid Howard, Shaoting Zhang, and Andrew Nealen. Hybrid mesh editing. In *Proceedings of EUROGRAPHICS Short papers*, pages 41–44, 2010.

[5] Mario Botsch, Mark Pauly, Martin Wicke, and Markus Gross. Adaptive space deformations based on rigid cells. *Computer Graphics Forum*, 26(3):339–347, 2007.

[6] Stephen Boyd. Relaxations and randomized methods for nonconvex QCQPs. *EE392o, Stanford University*, 2003.

[7] Matthew Brand and Aaron Hertzmann. Style machines. *ACM Transactions on Graphics (SIGGRAPH 2000)*, pages 183–192.

[8] Jinxiang Chai and Jessica K. Hodgins. Constraint-based motion optimization using a statistical dynamic model. *ACM Transactions on Graphics (SIGGRAPH 2007)*.

[9] Jinxiang Chai and Jessica K. Hodgins. Performance animation from low-dimensional control signals. *ACM Transactions on Graphics (SIGGRAPH 2005)*, pages 686–696.

[10] Stephen Chenney. Flow tiles. In *SIGGRAPH/Eurographics Symposium on Computer Animation '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 233–242, 2004.

[11] Myung Geol Choi, Manmyung Kim, Kyunglyul Hyun, and Jehee Lee. Deformable motion: Squeezing into cluttered environments. *Comput. Graph. Forum*, 30(2):445–453, 2011.

[12] Daniel Cohen-Or. Space deformations, surface deformations and the opportunities in-between. *J. Comput. Sci. Technol.*, 24(1):2–5, January 2009.

[13] Tim Davis. Umfpack version 5.6.1. http://www.cise.ufl.edu/research/sparse, 2012.

[14] Richard O Duda, Peter E Hart, and David G Stork. *Pattern classification*. Wiley-interscience, 2012.

[15] Wei-Wen Feng, Byung-Uck Kim, and Yizhou Yu. Real-time data driven deformation using kernel canonical correlation analysis. *ACM Transactions on Graphics (SIGGRAPH 2008)*, 27(3):1–9.

[16] Michael S. Floater. Mean value coordinates. *Computer Aided Geometric Design*, 20(1):19 – 27, 2003.

[17] Michael Gleicher. Retargeting motion to new characters. In *Proceedings of SIGGRAPH 98*, pages 33–42, 1998.

[18] Keith Grochow, Steven L. Martin, Aaron Hertzmann, and Zoran Popović. Style-based inverse kinematics. *ACM Transactions on Graphics (SIGGRAPH 2004)*, pages 522–531.

[19] Stephen. J. Guy, Jatin Chhugani, Changkyu Kim, Nadathur Satish, Ming Lin, Dinesh Manocha, and Pradeep Dubey. Clearpath: highly parallel collision avoidance for multi-agent simulation. In *Proceedings of the 2009 SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 177–187, 2009.

[20] Edmond S. L. Ho, Taku Komura, and Chiew-Lan Tai. Spatial relationship preserving character motion adaptation. *ACM Trans. Graph.*, 29(4):33:1–33:8, 2010.

[21] Kai Hormann and Michael S. Floater. Mean value coordinates for arbitrary planar polygons. *ACM Trans. Graph.*, 25:1424–1441, 2006.

[22] Haoda Huang, KangKang Yin, Ling Zhao, Yue Qi, Yizhou Yu, and Xin Tong. Detail-preserving controllable deformation from sparse examples. *IEEE Transactions on Visualization and Computer Graphics*, pages 1215–1227, 2012.

[23] Jin Huang, Xiaohan Shi, Xinguo Liu, Kun Zhou, Li-Yi Wei, Shang-Hua Teng, Hujun Bao, Baining Guo, and Heung-Yeung Shum. Subspace gradient domain mesh deformation. *ACM Trans. Graph.*, 25(3):1126–1134, 2006.

[24] Takeo Igarashi, Tomer Moscovich, and John F. Hughes. As-rigid-as-possible shape manipulation. *ACM Trans. Graph.*, 24:1134–1141, 2005.

[25] InterSense. Is-900 system. http://www.intersense.com.

[26] Alec Jacobson, Ilya Baran, Ladislav Kavan, Jovan Popović, and Olga Sorkine. Fast automatic skinning transformations. *ACM Trans. Graph.*, 31(4):77:1–77:10, 2012.

[27] Alec Jacobson, Ilya Baran, Jovan Popović, and Olga Sorkine. Bounded biharmonic weights for real-time deformation. *ACM Trans. Graph.*, 30(4):78:1–78:8, 2011.

[28] Kevin Jordao, Julien Pettré, Marc Christie, Marie-Paule Cani, et al. Crowd sculpting: A space-time sculpting method for populating virtual environments. *EUROGRAPHICS 2014*, 2013.

[29] Pushkar Joshi, Mark Meyer, Tony DeRose, Brian Green, and Tom Sanocki. Harmonic coordinates for character articulation. *ACM Trans. Graph.*, 26(3), 2007.

[30] Eunjung Ju, Myung Geol Choi, Minji Park, Jehee Lee, Kang Hoon Lee, and Shigeo Takahashi. Morphable crowds. *ACM Trans. Graph.*, 29(6):140:1–140:10, 2010.

[31] Tao Ju, Scott Schaefer, and Joe Warren. Mean value coordinates for closed triangular meshes. *ACM Trans. Graph.*, 24(3):561–566, 2005.

[32] Manmyung Kim, Youngseok Hwang, Kyunglyul Hyun, and Jehee Lee. Tiling motion patches. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 117–126, 2012.

[33] Manmyung Kim, Kyunglyul Hyun, Jongmin Kim, and Jehee Lee. Synchronized multi-character motion editing. *ACM Trans. Graph.*, 28:79:1–79:9, 2009.

[34] Lucas Kovar and Michael Gleicher. Automated extraction and parameterization of motions in large data sets. *ACM Trans. Graph.*, 23(3):559–568, 2004.

[35] Lucas Kovar, Michael Gleicher, and Frédéric Pighin. Motion graphs. *ACM Trans. Graph.*, 21(3):473–482, 2002.

[36] Björn Krüger, Jochen Tautges, Andreas Weber, and Arno Zinke. Fast local and global similarity searches in large motion capture databases. ACM SIGGRAPH/Eurographics Symposium on Computer Animation, pages 1–10, 2010.

[37] Taesoo Kwon, Kang Hoon Lee, Jehee Lee, and Shigeo Takahashi. Group motion editing. *ACM Trans. Graph.*, 27(3):80:1–80:8, 2008.

[38] Honglak Lee, Alexis Battle, Rajat Raina, and Andrew Y Ng. Efficient sparse coding algorithms. *Advances in neural information processing systems*, 19:801, 2007.

[39] Jehee Lee. Representing rotations and orientations in geometric computing. *IEEE Comput. Graph. Appl.*, pages 75–83, 2008.

[40] Jehee Lee, Jinxiang Chai, Paul S. A. Reitsma, Jessica K. Hodgins, and Nancy S. Pollard. Interactive control of avatars animated with human motion data. *ACM Transactions on Graphics (SIGGRAPH 2002)*, pages 491–500.

[41] Jehee Lee and Sung Yong Shin. A hierarchical approach to interactive motion editing for human-like figures. In *Proceedings of SIGGRAPH 99*, pages 39–48, 1999.

[42] Kang Hoon Lee, Myung Geol Choi, Qyoun Hong, and Jehee Lee. Group behavior from video: a data-driven approach to crowd simulation. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 109–118, 2007.

[43] Yoonsang Lee, Sungeun Kim, and Jehee Lee. Data-driven biped control. *ACM Trans. Graph. (SIGGRAPH 2010)*, 29:129:1–129:8.

[44] Alon Lerner, Yiorgos Chrysanthou, and Dani Lischinski. Crowds by example. *Computer Graphics Forum*, 26(3):655–664, 2007.

[45] Alon Lerner, Eitan Fitusi, Yiorgos Chrysanthou, and Danny Cohen-Or. Fitting behaviors to pedestrian simulations. *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 199–208, 2009.

[46] Yaron Lipman, Johannes Kopf, Daniel Cohen-Or, and David Levin. GPU-assisted positive mean value coordinates for mesh deformations. In *Proceedings of the fifth Eurographics symposium on Geometry processing*, pages 117–123, 2007.

[47] Yaron Lipman, David Levin, and Daniel Cohen-Or. Green coordinates. *ACM Trans. Graph.*, 27:78:1–78:10, 2008.

[48] Yaron Lipman, Olga Sorkine, David Levin, and Daniel Cohen-Or. Linear rotation-invariant coordinates for meshes. *ACM Trans. Graph.*, 24(3):479–487, 2005.

[49] Guodong Liu, Jingdan Zhang, Wei Wang, and Leonard McMillan. Human motion estimation from a reduced marker set. ACM Symposium on Interactive 3D graphics and games, pages 35–42, 2006.

[50] Huajun Liu, Xiaolin Wei, Jinxiang Chai, Inwoo Ha, and Taehyun Rhee. Realtime human motion control with a small number of inertial sensors. In *Symposium on Interactive 3D Graphics and Games*, pages 133–140, 2011.

[51] Ligang Liu, Lei Zhang, Yin Xu, Craig Gotsman, and Steven J. Gortler. A local/global approach to mesh parameterization. In *Proceedings of Eurographics Symposium on Geometry Processing*, volume 27, pages 1495–1504, 2008.

[52] Thomas Melzer, Michael Reiter, and Horst Bischof. Appearance models based on kernel canonical correlation analysis. *Pattern Recognition*, 36(9):1961 – 1971, 2003.

[53] David M. Mount and Sunil Arya. Library for approximate nearest neighbor searching. http://www.cs.umd.edu/ mount/ANN, 2010.

[54] Tomohiko Mukai and Shigeru Kuriyama. Geostatistical motion interpolation. *ACM Trans. Graph.*, 24(3):1062–1070, 2005.

[55] Meinard Müller, Tido Röder, and Michael Clausen. Efficient content-based retrieval of motion capture data. In *ACM Transactions on Graphics (TOG)*, pages 677–685, 2005.

[56] Soraia R. Musse and Daniel Thalmann. A model of human crowd behavior: Group inter-relationship and collision detection analysis. In *Computer Animation and Simulation '97*, pages 39–51, 1997.

[57] Rahul Narain, Abhinav Golas, Sean Curtis, and Ming C. Lin. Aggregate dynamics for dense crowd simulation. *ACM Trans. Graph.*, 28(5):122:1–122:8, 2009.

[58] Sachin Patil, Jur Van Den Berg, Sean Curtis, Ming C Lin, and Dinesh Manocha. Directing crowd simulations using navigation fields. *IEEE Transactions on Visualization and Computer Graphics*, 17(2):244–254, 2011.

[59] PCL. Point Cloud Library. http://pointclouds.org/, 2012.

[60] N. Pelechano, J. M. Allbeck, and N. I. Badler. Controlling individual agents in high-density crowd simulation. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 99–108, 2007.

[61] N. Pelechano, K. O'Brien, B. Silverman, and N. Badler. Crowd simulation incorporating agent psychological models, roles and communication. In *Proceedings of the First International Workshop on Crowd Simulation*, pages 24–25, 2005.

[62] Craig W. Reynolds. Flocks, herds and schools: A distributed behavioral model. In *Proceedings of SIGGRAPH 87*, pages 25–34, 1987.

[63] Alla Safonova and Jessica K. Hodgins. Construction and optimal search of interpolated motion graphs. *ACM Trans. Graph.*, 26(3), 2007.

[64] Jonathan Richard Shewchuk. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In *Applied Computational Geometry: Towards Geometric Engineering*, volume 1148 of *Lecture Notes in Computer Science*, pages 203–222. Springer-Verlag, May 1996.

[65] Hyun Joon Shin and Jehee Lee. Motion synthesis and editing in low-dimensional spaces: Research articles. *Comput. Animat. Virtual Worlds*, pages 67 – 94, 2006.

[66] Hyun Joon Shin, Jehee Lee, Sung Yong Shin, and Michael Gleicher. Computer puppetry: An importance-based approach. *ACM Transactions on Graphics*, pages 67–94, 2001.

[67] Ronit Slyper and Jessica Hodgins. Action capture with accelerometers. ACM SIGGRAPH / Eurographics Symposium on Computer Animation, pages 193–199, 2008.

[68] Olga Sorkine and Marc Alexa. As-rigid-as-possible surface modeling. In *Proceedings of the fifth Eurographics symposium on Geometry processing*, pages 109–116, 2007.

[69] Olga Sorkine, Daniel Cohen-Or, Yaron Lipman, and Marc Alexa. Laplacian surface editing. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Geometry Processing*, pages 175–184, 2004.

[70] Jos F. Sturm. Using sedumi 1.02, a matlab toolbox for optimization over symmetric cones. *Optimization Methods and Software*, 11(1-4):625–653, 1999.

[71] Robert W. Sumner, Johannes Schmid, and Mark Pauly. Embedded deformation for shape manipulation. *ACM Trans. Graph.*, 26(3):80, 2007.

[72] Jochen Tautges, Arno Zinke, Björn Krüger, Jan Baumann, Andreas Weber, Thomas Helten, Meinard Müller, Hans-Peter Seidel, and Bernd Eberhardt. Motion reconstruction using sparse accelerometer data. *ACM Transactions on Graphics*, pages 18:1–18:12, 2010.

[73] Adrien Treuille, Seth Cooper, and Zoran Popovic. Continuum crowds. *ACM Trans. Graph.*, 25(3):1160–1168, 2006.

[74] Nobuyuki Umetani, Danny Kaufman, Takeo Igarashi, and Eitan Grinspun. Sensitive Couture for Interactive Garment Editing and Modeling. *ACM Trans. Graph.*, 30(4), 2011.

[75] Xiaolin Wei and Jinxiang Chai. Intuitive interactive human character posing with millions of example poses. *IEEE Computer Graphics and Applications*, pages 78–88, 2009.

[76] Weiwei Xu, Kun Zhou, Yizhou Yu, Qifeng Tan, Qunsheng Peng, and Baining Guo. Gradient domain editing of deforming mesh sequences. *ACM Trans. Graph.*, 26(3), 2007.

[77] KangKang Yin and Dinesh K. Pai. Footsee: an interactive animation system. ACM SIGGRAPH / Eurographics Symposium on Computer Animation, pages 329–338, 2003.

초 록

사실적인 캐릭터 애니메이션을 만들기 위하여 캐릭터 합성 및 편집에 관련된 많은 애니메이션 시스템들이 제안되었다. 하지만, 기존의 애니메이션 시스템을 이용하여 고품질의 캐릭터 애니메이션을 만들기 위해서는 애니메이터들의 높은 숙련도가 필요한 실정이다. 따라서 보다 쉽고 간편하게 캐릭터 애니메이션을 합성 및 편집할 수 있는 새로운 애니메이션 시스템이 요구되고 있다.

효율적인 애니메이션 시스템을 위해 고려해야 할 사항이 크게 두 가지가 있다. 첫째, 직관적으로 캐릭터들을 조절 할 수 있어야 한다. 일반적인 하나의 캐릭터 모델에 대해서 조절해야 할 관절 정보가 많다. 또한, 여러 명의 캐릭터에 대해서도 사람 숫자가 많아지거나 환경이 변함에 따라 캐릭터-캐릭터 혹은 캐릭터-환경과의 충돌 처리와 같이 조절해야 될 것들이 많아지게 된다. 둘째, 사용자와 애니메이션 시스템 간의 실시간 인터랙션이 필요하다. 캐릭터 애니메이션은 복잡하기 때문에 원하는 결과 애니메이션을 만들기까지 사용자는 컴퓨터와 반복적인 인터랙션을 수행하게 된다. 이 과정에서 빠른 피드백을 받으며 작업하기 위한 실시간 인터랙션이 필요하다. 하지만, 이러한 것들을 고려한 애니메이션 시스템을 구축하는 것은 쉽지 않다. 사용자가 주로 이용하는 컴퓨터나 모션 센서 등에서 얻어올 수 있는 시스템 입력 값들을 낮은 차원 정보 인데 비해, 캐릭터 애니메이션은 높은 차원 정보를 담고 있기 때문이다.

본 학위 논문의 목표는 낮은 차원의 사용자 입력으로부터 높은 차원의 캐릭터 애니

메이션의 인터랙티브한 합성과 편집을 가능하도록 하는 것이다. 이를 위해 사용자 입력과 시스템 출력의 연결고리를 찾아야 한다. 본 학위 논문에서는 시스템 입력으로부터 캐릭터 애니메이션을 생성하는 데이터 기반 매핑 모델을 제안한다. 모션 데이터를 이용하면, 시스템 입력의 모자란 정보를 모션 데이터가 가지고 있는 정보로부터 추론할 수가 있고 모션데이터는 실제 사람 동작의 디테일을 가지고 있기 때문에 이를 통해 사실적인 캐릭터 애니메이션을 만들 수 있다. 본 학위 논문에서는 모션 캡쳐 데이터, 시뮬레이션 데이터, 그리고 다른 모션 합성 알고리즘으로부터 얻어온 데이터를 이용하여 데이터 기반 매핑 모델을 확립하였다.

두 가지 애니메이션 시스템을 통해 데이터 기반 매핑 모델에 기반한 캐릭터의 합성 및 편집의 효율성을 입증하였다. 첫 번째 시스템에서는 3차원 모션 센서로부터 사용자 자세를 연결해주는 리얼타임 매핑 모델을 확립하고 이를 통해, 사용자의 움직임을 나타내는 사실적인 캐릭터 애니메이션이 생성하였다. 두 번째 시스템에서는 수백 명의 캐릭터들이 주어져 있을 때, 캐릭터들의 이동 경로를 자유자재로 변화시킬 수 있는 매니퓰레이션 매핑 모델을 확립하고 이를 통해, 새로운 환경에 적합한 군중 애니메이션을 생성하였다.

주요어: 컴퓨터 그래픽스, 인간 동작, 데이터 기반 애니메이션, 성능 애니메이션, 인터랙티브 편집, 기계학습, 수치 최적화

학번: 2007-20955